Tablica koncepcyjna

Autorzy: Łukasz Klasiński, Marcin Witkowski

Dlaczego tu jesteśmy?

→ Brak alternatyw dla już przestarzałego oprogramowania (Picassa3)

→ GNU/Linux'owe odpowiedniki są słabe (UIX, face-recognition)

→ W dobie cyfryzacji przydałaby się realny odpowiednik "analogowych albumów"

Krótkie podsumowanie

→ Program dla każdego, kto ma dosyć losowo porozrzucanych zdjeć

→ Osób, które lubią porządek

→ Mają dosyć niedopracowanych konkurencyjnych rozwiązań

Rozwiązanie:

PhotoCHAD

PHOTOCHAD

Bazowe funkcjonalności

- → Łatwa manipulacja lokalnymi zbiorami
- → Możliwość szybkiego otagowania zdjęć ze względu na osoby się na nim znajdujące..
- → ..ale także ich ilość, czas, lokalizacje czy nawet ustawienia obiektywu (np. zdjęcia macro)
- → Ustandaryzowany sposób kopiowania zdjęć z różnych nośników (aparaty, smartphone, etc.)
- → Automatyczne wykrywanie duplikatów

Czego nie chcemy

- → Kolejnego programu do edycji rastorowej (nie wyklucza to jednak ew. prostych edycji jak filtry)
- → Oprogramowania ze słabym interfejsem graficznym - UIX jest tak samo ważne jak backend
- → Braku wsparcia dla najważniejszych platform

Ewentualny plany na przyszłość

- Integracja z portalami społecznościowymi (Facebook, Instagram, Flickr, Google Photos, etc.)
- Proste edycje zdjęć jak color grading, skalowanie, konwersja do innych formatów czy czyszczenie danych EXIF
- ...

Do kogo kierujemy nasz produkt

→ Profesjonalistów (fotografowie)

→ Pasjonatów zdjęć

→ Zwykłych użytkowników

→ Studentów (zdjęcia notatek, wykładów, media społecznościowe)

Architektura i technologie

→ Rust + winit

→ SQLite

→ TensorFlow + FaceNet / OpenFace

→ Ew. rozszerzalność w Lua

Największe wyzwania i możliwe problemy

- → *Dobry* design aplikacji
- → «Stabilna» metoda rozpoznawania twarzy i obiektów
- → Rust jest relatywnie młodym językiem, a jego biblioteki nie do końca dojrzałe – możliwe bugi
- → Odporność na uszkodzenia bazy danych
- → Wydajny program, który działa w tle bez obciążenia systemu

Ramy Czasowe

→ Wytwarzanie 9 miesięcy

→ Testowanie 3 miesiące

→ Wdrożenie 3 miesiące

Elastyczne Dostosowanie

- → Możemy początkowo pominąć optymalizację i testy na mniej znaczących platformach, tnąc koszty i czas (np. Linux)
- → Skorzystanie z wolniejszych, ale pewniejszych rozwiązań (np. C#, OpenCV) zmniejszając czas potrzebny testy

Kogo potrzebujemy?

→ 2 senior programistów Rust oraz 2 junior

→ 1 specjalistę baz danych

→ 1 programistę aplikacji mobilnych

→ 2 specjalistów machine learning

→ 2 testerów

Ile to będzie kosztowało

- Junior Rust developer 50zł/h
- Senior Rust developer 70zł/h
- Specjalista baz danych 120zł/h
- Specjalista machine learning 130zł/h
- Tester 60zł/h

Koszt zamortyzowany: (50*2 + 70*2 + 120 + 130*2) *
5 * 8 * 4 * 12 + (2 * 60 * 8 * 4 * 3) ~ 1mln zł

Co należy zrobić

→ Projekt UI, opisujący dodatkowo wszystkie bazowe funkcjonalności (ich wygląd, kontrolki, etc.)
→ Projekt architektury bazy danych
→ Bazową, abstrakcyjną implementacje GUI
→ Stworzenie testów a następnie implementacje poszczególnych funkcjonalności aplikacji jako osobne moduły
→ Połączenie 'backendu z frontendem'
→ Betatesty pod względem użytkowania oprogramowania w zamkniętej grupie użytkowników
→ Ostateczne szlify i poprawki (zawsze są jakieś..)