Daniil Yuzepenko
+48790803659

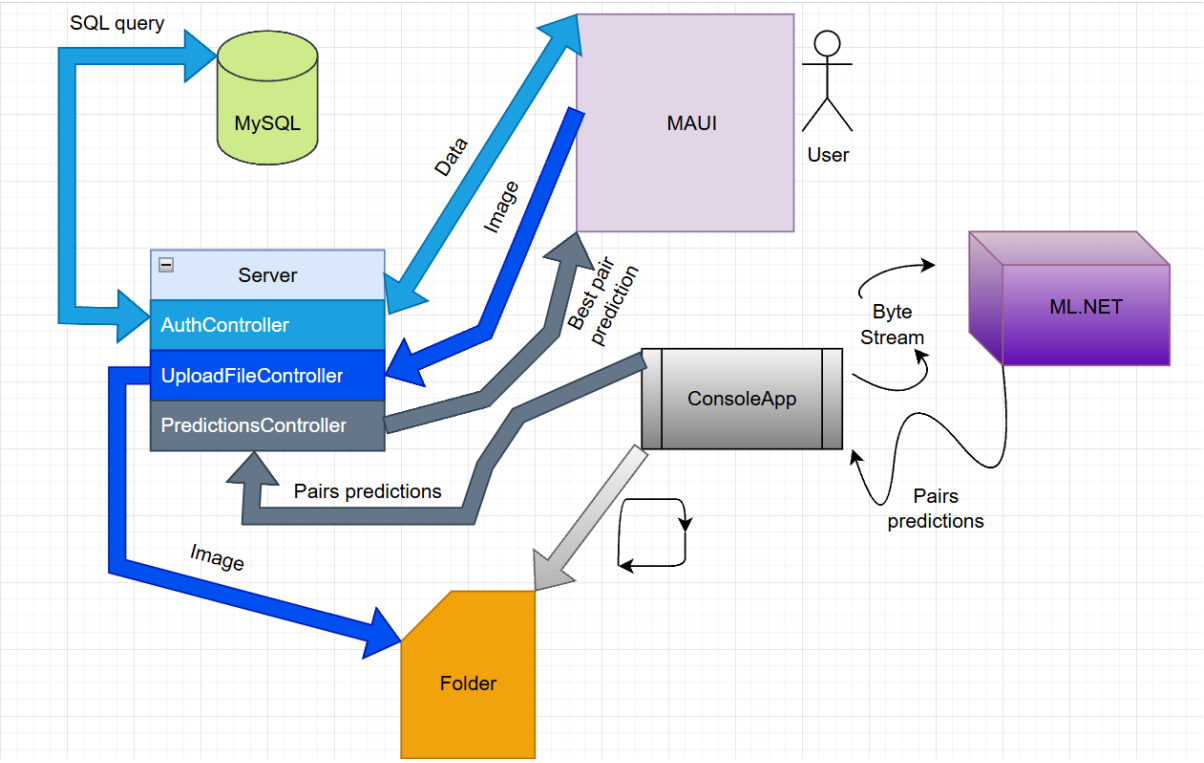
georgiferstor@gmail.com

PriceGO

Projekt jest w pełni napisany w C# (MAUI.NET + ASP.NET + ML.NET) i korzysta z bazy danych MySQL. Temat projektu jest bardzo podobny do Google Lens – aplikacji, która przetwarza zdjęcia różnych urządzeń technicznych, analizuje, co na nich jest przedstawione, i zwraca wyniki (predict).

Aplikacja została napisana przy użyciu technologii MAUI, serwer działa na ASP.NET, a także wykorzystano model ML.NET.

Schemat działania programu



Serwer: Serwer działa w lokalnej sieci Hot spot z komputera, co pozwala na dostęp do serwera ze wszystkich urządzeń podłączonych do komputera. Sam serwer zawiera trzy WEB.API, które zostały zaimplementowane w postaci kontrolerów:

- **AuthController.cs** – dla systemu logowania i rejestracji nowych użytkowników;
- **UploadFileController.cs** – do wysyłania zdjęć z telefonu na komputer, gdzie znajduje się serwer;

- **PredictionsController.cs** – do uzyskiwania wyników z programu konsolowego i przekazywania wyników do aplikacji MAUI.

MAUI: Aplikacja zawiera system logowania i rejestracji. Aplikacja wysyła zapytania GET do serwera (AuthController), który następnie wykonuje zapytanie SQL do bazy danych MySQL, po czym zwraca wyniki do programu. W zależności od wyników logujemy się lub nie, a także sprawdzamy, czy rejestracja przebiegła pomyślnie. Po udanym logowaniu trafiamy na stronę główną aplikacji, gdzie znajduje się moduł kamery (CameraView) i dwa przyciski do komunikacji: Zrób zdjęcie lub wybierz zdjęcie z galerii:

1. **Z galerii** Aby wybrać zdjęcie z galerii (UploadImage_Clicked), wybieramy je za pomocą MediaPicker, zapisujemy zdjęcie w katalogu cache, ustawiamy nazwę pliku i ścieżkę pliku naszego zdjęcia, a następnie wywołujemy metodę Send_to_API(). Ta metoda nie wysyła bezpośrednio zdjęcia do API, szczegóły będą opisane później.
2. **Zrób zdjęcie** Na stronie wyświetlana jest kamera. Jest również przycisk, po naciśnięciu którego (Take_photo_Clicked) odwołujemy się do kamery (await Camera.CaptureImage(CancellationToken.None)), robimy zdjęcie, które kamera uchwyciła w momencie naciśnięcia przycisku. Zdarzają się opóźnienia między naciśnięciem przycisku a momentem, w którym obraz zostaje skopiowany do e.Media. Po zrobieniu zdjęcia zapisujemy je w katalogu cache, ustawiamy nazwę pliku i ścieżkę pliku. Aby zobaczyć wynik, wyświetlamy zdjęcie na urządzeniu. Następnie, w metodzie przycisku (Take_photo_Clicked), wywołujemy metodę Send_to_API().

Send_to_API(): Metoda ta działa na podstawie dwóch pól: fileName i filePath. Używając tych pól, otwieramy zdjęcie w postaci strumienia pliku (fileStream) i wysyłamy je do API za pomocą metody UploadFileAsync z klasy mainPage_UploadFile_API. Metoda ta jest już bezpośrednio połączona z UploadFileController i wysyła zdjęcie. Po tym wywołujemy metodę Predictions z klasy predictions_web_API. Ta klasa jest bezpośrednio połączona z PredictionsController.cs.

Następnie UploadFileController.cs po pomyślnym przetworzeniu zapytania POST na serwerze zapisuje zdjęcie w katalogu na komputerze (serwerze) w podanej ścieżce.

Ten katalog jest na bieżąco monitorowany przez aplikację konsolową (ML_Tech_ConsoleApp1), która ciągle sprawdza dodanie nowych plików. Gdy tylko nowy plik zostanie dodany, aplikacja konsolowa bierze nazwę pliku, łączy pełną ścieżkę do pliku i zapisuje ją jako strumień bajtów. Następnie tworzony jest obiekt różnych wbudowanych klas z ML.NET, które przyjmują ten strumień i dalej go przetwarzają. Nie udało mi się znaleźć otwartego kodu matematycznych algorytmów ML.NET, ale przeczytałem, że model ten wygładza obraz, następnie pozostawia tylko kontury, a na podstawie tych konturów i etykiet (które model nauczył się rozpoznawać w trakcie treningu) otrzymujemy wyniki dla każdej klasy (PredictAllLabels). Wyniki te są wyświetlane w konsoli, a następnie przekazywane do PredictionsController.cs przez POST za pomocą metody SendPredictionsToApi(). Następnie nasz API PredictionsController wybiera najlepsze wartości z par klucz/wartość i zapisuje je.

Po tym wszystkim, aplikacja MAUI wysyła zapytanie GET do PredictionsController, aby uzyskać wyniki, ale zapytanie GET zostanie wykonane dopiero po zakończeniu zapytania POST, dzięki czemu unikamy problemu, że aplikacja konsolowa jeszcze nie zdążyła przekazać wyników, a aplikacja MAUI już próbuje wykonać zapytanie GET.

Program jeszcze nie jest w pełni ukończony, ponieważ pierwotnie planowano, że na podstawie rozpoznanych obiektów na zdjęciu będzie sugerować różne opcje zakupu w internecie (np. z MediaExpert, MediaMarkt lub nawet połączyć ją ze swoim własnym sklepem internetowym). Niestety, nie zdążyłem jeszcze tego zaimplementować. Dlatego nazwa PriceGo :)

Jeśli chodzi o główne trudności napotkane podczas tworzenia projektu, to były to:

1. Próby połączenia ML.NET bezpośrednio z systemem Android, bez użycia serwera. Ponadto w internecie prawie nie ma przykładów ani informacji na temat integracji ML.NET z Androidem. Istnieją tylko materiały dotyczące użycia innych modeli ML.NET. Ze względu na brak wiedzy o tym, co dokładnie dzieje się w samej strukturze ML.NET, przez długi czas nie mogłem znaleźć wyjątku, który bez żadnego opisu zatrzymywał moją aplikację, gdy tylko próbowałem wysłać zdjęcie do modelu. Co ciekawe, na systemie Windows ten problem nie występował, mimo że używałem dokładnie tego samego kodu. Dzięki logom i pomocy wykładowcy z Politechniki udało mi się zrozumieć, że wyjątek ten był generowany przez **MonoDebugger**, ponieważ konfliktował z **AndroidDebugger**.
2. Problemy z kamerą – początkowo używałem wbudowanej biblioteki **Camera.MAUI**, ale po wielu próbach jej zrozumienia doszedłem do wniosku, że lepiej skorzystać z zewnętrznej biblioteki **CameraView**.
3. Różnice w sposobie działania **MediaPicker** i **e.Media** z **CameraView** przy przetwarzaniu obrazów. **MediaPicker** korzysta ze **stream**, a **e.Media** używa **fileStream**.
4. Problemy z wyjątkami podczas korzystania z **CameraView**. Nadal do końca ich nie rozwiązałem, ale „metodą prób i błędów” udało mi się znaleźć obejście. Problem polegał na tym, że podczas wywoływania metody przechwytywania zdjęcia nie można było jej oznaczyć jako **asynchronicznej**, ponieważ powodowało to wyjątek na platformie Android, którego nie udało mi się przechwycić za pomocą **try-catch**.

P.S Przepraszam, jeśli jest za dużo tekstu.