



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Informatyki

Praca inżynierska

*Aplikacja do optymalizacji przepływu pojazdów na skrzyżowaniu
poprzez zmiany cykli świateł ulicznych*

*Application for optimizing traffic flow at crossroads
by controlling traffic lights*

Autorzy:	Nikodem Korohoda, Mateusz Więcek, Jędrzej Ziebura, Szymon Żychowicz
Kierunek studiów:	Informatyka
Opiekun pracy:	dr hab. inż. Paweł Topa, prof. AGH

Kraków, 2025

Spis treści

1	Cel prac i wizja produktu	5
1.1	Opis dziedziny problemu	6
1.2	Charakterystyka problemu	7
1.3	Cele projektu	8
1.4	Porównanie z podobnymi rozwiązaniami	10
1.4.1	PTV Group	10
1.4.2	Krakowski Obszarowy System Sterowania Ruchem (UTSC)	11
1.4.3	Isarsoft	12
1.5	Główne funkcje systemu	13
1.6	Główne założenia technologiczne	14
1.7	Analiza ryzyka	15
2	Zakres funkcjonalności	17
2.1	Użytkownicy systemu	17
2.1.1	Specjalista ruchu drogowego	17
2.1.2	Użytkownik z dostępem do nagrań ruchu drogowego	17
2.1.3	Administrator	17
2.2	Zewnętrzne systemy współpracujące	18
2.2.1	Docker	18
2.2.2	Maven	18
2.2.3	Spring	18
2.2.4	Testcontainers	18
2.2.5	Mockito	19
2.2.6	MongoDB i GridFS	19
2.2.7	FastAPI	19
2.2.8	OpenCV	19
2.2.9	CBC	19
2.2.10	Gecode	20
2.2.11	Google Maps	20
2.2.12	React	20
2.2.13	Styled-components i Material UI	20
2.3	Wymagania	21
2.3.1	Wymagania funkcjonalne	21
2.3.2	Wymagania нефункционалне	28
2.4	Scenariusze użytkowania	30
2.4.1	Tworzenie skrzyżowania	31
2.4.2	Dodawanie pierwotnej sekwencji sygnalizacji świetlnej	35
2.4.3	Dodawanie nagrania	36

2.4.4	Zlecenie optymalizacji	38
2.4.5	Podgląd najnowszej optymalizacji	40
3	Wybrane aspekty realizacji	41
3.1	Frontend	42
3.1.1	Stos technologiczny	42
3.1.2	Zaawansowane mechanizmy modułu	43
3.1.3	Problemy techniczne	48
3.1.4	Aspekt graficzny	49
3.2	Backend	49
3.2.1	Stos technologiczny	49
3.2.2	Architektura	49
3.2.3	Komponenty	50
3.2.4	Testowanie	54
3.3	Baza Danych	56
3.3.1	Stos technologiczny	56
3.3.2	Architektura i implementacja	56
3.3.3	Schemat Bazy Danych	57
3.4	Optymalizator	58
3.4.1	Serwer Optymalizatora	58
3.4.2	Optymalizacja	59
3.5	Analizator	63
3.5.1	Serwer Analizatora	63
3.5.2	Wykrywanie pojazdów	64
4	Organizacja pracy	65
4.1	Charakterystyka projektu	65
4.2	Podział zadań	67
4.3	Wykorzystane narzędzia	68
4.3.1	Komunikacja	68
4.3.2	Współpraca	68
4.3.3	Przebieg pracy	70
5	Wyniki projektu	71
5.1	Uzyskany produkt	71
5.2	Efekty działania aplikacji	72
5.3	Ograniczenia rozwiązania	76
5.4	Propozycje dalszego rozwoju	77
5.5	Podsumowanie	77
	Spis rysunków	80
	Spis tabel	82

Rozdział 1

Cel prac i wizja produktu

Celem pracy jest stworzenie produktu, dzięki któremu klient będzie mógł zaplanować efektywny system sterujący cyklami zmian świateł ulicznych. Pozwoli on użytkownikowi zoptymalizować przepływ ruchu drogowego, co niesie pozytywne skutki na wielu płaszczyznach. Problem jest bardzo powszechny, bo dotyczy niemal każdego miasta.

Aplikacja, po dostarczeniu przez użytkownika za pomocą interfejsu graficznego informacji dotyczących układu skrzyżowania oraz nagrań ruchu ulicznego, przeanalizuje je, wykorzystując model uczenia maszynowego i pozyska z nich potrzebne dane. Zostaną one następnie przekazane do modelu opierającego się na programowaniu z ograniczeniami, który z wykorzystaniem solvera zaproponuje optymalizację. Jej rezultat będzie można wyświetlić w formie danych opisowych albo symulacji wizualnej.



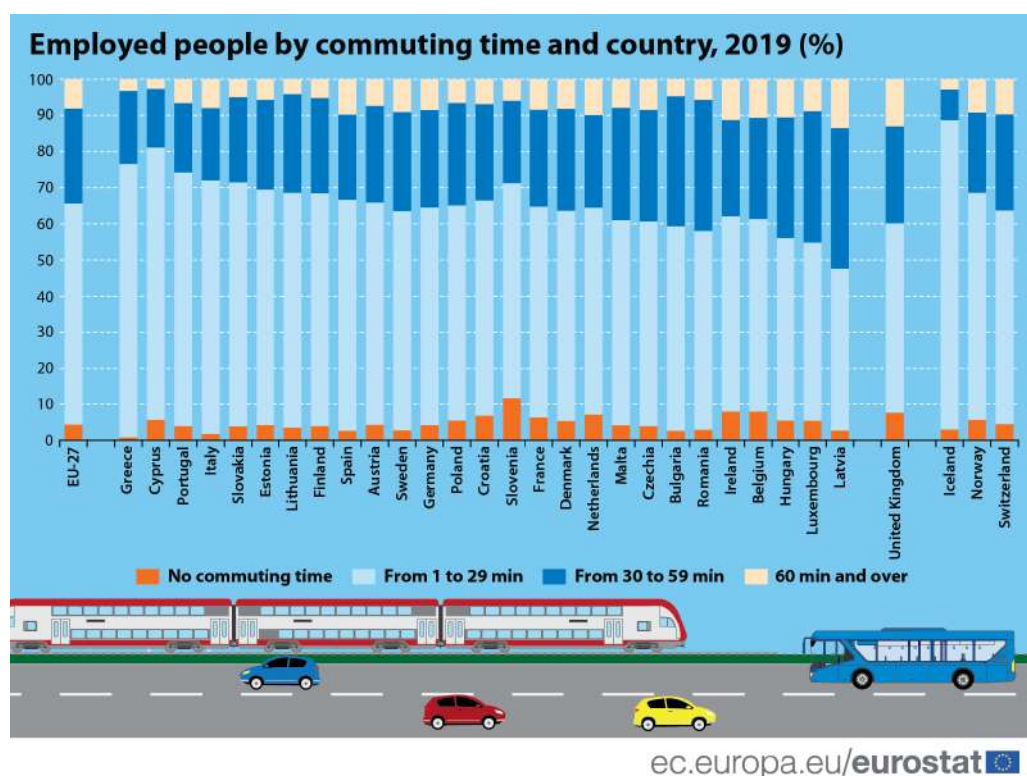
Rysunek 1.1: Logo aplikacji TrafficFlowOptimizer

Produkt nazwano TrafficFlowOptimizer. W dalszej części pracy będzie również określany skrótem TFO (patrz rys. [1.1](#)).

1.1. Opis dziedziny problemu

W obecnych czasach znaczna część społeczeństwa porusza się wykorzystując samochody i inne pojazdy zmotoryzowane. Według danych Eurostatu [1] w 2019 roku zatrudnieni Polacy poświęcali średnio 24 minuty dziennie na dojazd do pracy w jedną stronę (patrz rys. 1.2). Niecałe 35% z nich poświęcało na podróż tam i z powrotem co najmniej godzinę dziennie, a prawie co czwarty od godziny do półtorej. W miastach było to odpowiednio 43,1% oraz 31,5% pracujących.

Problem nie dotyczy wyłącznie Polski. Średni czas dojazdu Europejczyków do pracy jest zbliżony i wynosi 25 minut w jedną stronę. Ponad 35% z nich potrzebuje co najmniej godziny dziennie na dotarcie i powrót z pracy, a więcej niż 20% od godziny do półtorej.



Rysunek 1.2: Czas dojazdu do pracy w jedną stronę w europejskich krajach

Źródło grafiki: Eurostat [2]

Niezależnie od tego czy korzystamy z transportu publicznego, czy prywatnego jesteśmy narażeni na zatory uliczne, czyli tzw. korki. Oprócz negatywnych efektów bezpośrednio ingerujących w życie przeciętnego obywatela tj. stres, frustracja, spóźnienia, koszty paliwa lub biletów niosą one także niekorzystne społeczne i globalne skutki, np. zanieczyszczenie powietrza, zanieczyszczenie hałasem i ich konsekwencje zdrowotne [3, 4], a także globalne ocieplenie [5] oraz niebezpieczeństwo na drodze.

Inteligentne systemy sterowania ruchem są implementowane w wielu krajach na całym świecie. Wykorzystywane jest wiele metod i podejść, które umożliwiają zwiększenie płynności ruchu, a co za tym idzie – redukcję wyżej wymienionych szkód.

1.2. Charakterystyka problemu

Istnieje wiele czynników powodujących korki. Część z nich nie jest zależna od organizacji ruchu, wśród nich między innymi warunki pogodowe lub wypadki drogowe. Jednocześnie niektóre problemy można rozwiązać, na przykład sposób organizacji remontów. Jednak w opinii zespołu główną przyczyną korków są nieoptymalne cykle zmian świateł, które powodują zatory na skrzyżowaniach. Za cel pracy obrano zatem stworzenie aplikacji pozwalającej na usprawnienie przejazdów przez skrzyżowania uliczne dzięki dostosowywaniu zmian świateł do natężenia ruchu.

Propozycja rozwiązania problemu zagęszczeń ruchu przedstawiona przez zespół będzie opierała się przede wszystkim na informacjach o przewidywanych i uśrednionych liczebnościach pojazdów nadjeżdżających z poszczególnych ulic w kierunku skrzyżowania. Dzięki temu możliwe będzie zapewnienie ich usprawnionego przepływu. Przykładowo w godzinach porannych prawdopodobnie więcej pojazdów będzie zmierzało w kierunku centrum (patrz rys. 1.3), wieczorem natomiast, w kierunku obrzeży miasta, zatem cykle świateł również powinny być zależne od pory dnia.



Rysunek 1.3: Przykład wystąpienia powyżej opisanej dysproporcji ruchu

Londyn, 2023

Niestety problem zmiany świateł na skrzyżowaniu nie jest problemem trywialnym. Należy bowiem zapewnić zgodność ze wszystkimi zasadami ruchu drogowego, takimi jak przejazdy na światłach kolizyjnych oraz bezkolizyjnych, korzystanie z przejazdów oznaczonych zieloną strzałką w prawo oraz inne [6].

1.3. Cele projektu

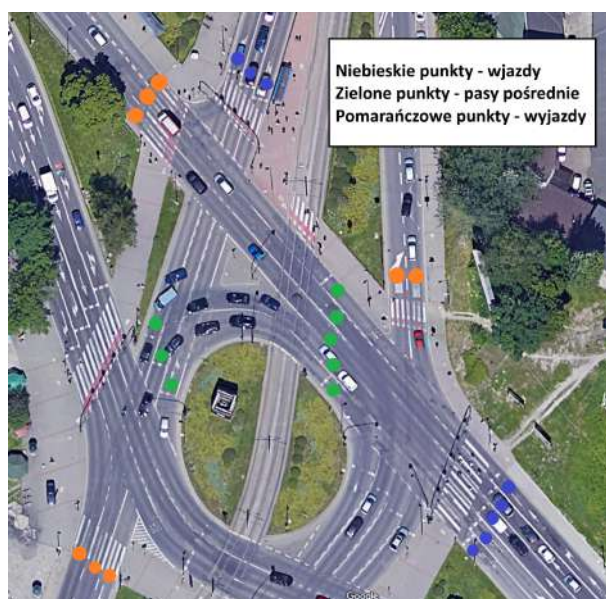
Celem pracy jest stworzenie aplikacji usprawniającej komunikację na skrzyżowaniach, funkcjonującej niezależnie od innych podobnych serwisów lub narzędzi. Będzie ona opierała się na technologii webowej, a co za tym idzie, jej użytkownik będzie w stanie korzystać z niej z dowolnej przeglądarki bez konieczności instalowania plików źródłowych.

Celem projektu jest również stworzenie jej w taki sposób, aby bez problemu można było zastosować ją do poprawy ruchu zróżnicowanych skrzyżowań, podając odpowiednie parametry opisane bardziej szczegółowo poniżej. Nie będzie ona wymagała wiedzy specjalistycznej z dziedziny optymalizacyjnej ani informatycznej, a jedynie poprawnego zainicjalizowania skrzyżowania i dostarczenia odpowiednich nagrań.

TFO będzie proponował usprawnienia przejazdu pojazdów przez skrzyżowania, za pomocą zmian cykli występującej na nim sygnalizacji świetlnej. Zostanie to osiągnięte za pomocą solvera, który dokona optymalizacji zmian świateł ulicznych tak, aby zmieniały się one adekwatnie do natężenia ruchu. Informacje na temat rzeczywistego przepływu pojazdów będzie można dostarczyć do aplikacji za pomocą nagrań ruchu ulicznego — będą one następnie analizowane przez system wykrywania pojazdów wykorzystujący konwolucyjną sieć neuronową [7] oraz konwertowane na dane liczbowe określające ile samochodów pojawia się średnio na minutę na danym pasie każdej z ulic. Aplikacja nie będzie zapewniała narzędzia do pozyskiwania owych nagrań, ich zdobycie leży w gestii użytkownika. Będzie on również musiał określić, w jakich godzinach dane nagranie zostało zarejestrowane i którego z pasów dotyczy.

Użytkownik na etapie konfiguracji będzie mógł wybrać jedno z zainicjalizowanych już skrzyżowań lub wprowadzić wybrane przez siebie poprzez kolejno:

1. wybranie z Map Google fragmentu, na którym znajduje się szukane skrzyżowanie
2. zaznaczenie, gdzie oraz ile jest pasów wjazdowych, wyjazdowych czy pośrednich na skrzyżowaniu, przy czym jako pas pośredni rozumiany jest pas z sygnalizacją, niebędący wjazdem na skrzyżowanie (przykład, patrz rys. 1.4)



Rysunek 1.4: Przykładowe skrzyżowanie z zaznaczonymi niektórymi pasami zgodnie z instrukcją

3. określenie, z którego pasa wjazdowego można dostać się do którego pasa wyjazdowego
4. zaznaczenie, gdzie oraz jakie światła znajdują się na tym skrzyżowaniu
5. określenie, które przejazdy są względem siebie kolizyjne, czyli samochody nie mogą przez nie przejeżdżać równocześnie

Kolejne kroki będą realizowane poprzez kliknięcia w odpowiednie miejsce na ekranie i wypełnianie niewielkiego formularza, a każda zmiana modelu skrzyżowania będzie wizualizowana na bieżąco.

Podczas użytkowania, po każdorazowym dostarczeniu nagrania, TFO wykorzysta algorytmy służące do wykrywania ruchu oraz wygeneruje dane określające jak często pojazdy pojawiają się na danych pasach ulic oraz w którą stronę zmierzają w godzinach, kiedy nagranie zostało zarejestrowane. Po uzyskaniu tych danych użytkownik będzie mógł zlecić optymalizację. Dokona się ona za pomocą solvera, który zachowując wszystkie konieczne założenia, m.in. zapewnienie braku kolizyjności niektórych przejazdów, dąży do zmaksymalizowania wartości funkcji celu. Najważniejszym wymaganiem, które powinno być zaspokojone, jest proporcjonalność czasu, przez jaki dane połączenie na skrzyżowaniu jest przejezdne w stosunku do przewidywanej liczby samochodów. Ma to za zadanie doprowadzić do balansu względem tendencji ruchu drogowego.

Ponieważ solver opiera się na uśrednionej liczbie samochodów oraz działa w trybie offline względem aktualnego ruchu drogowego, nie jest możliwe zapewnienie idealnego dopasowania do warunków na ulicy w danym momencie. Z tego powodu zdecydowano się na optymalizację proporcjonalności.

Możliwe będzie zlecenie odnalezienia optymalnego rozwiązania, jednak ponieważ takie poszukiwanie może być bardzo czasochłonne, aplikacja pozwoli również na znalezienie nieoptymalnego rozwiązania, jednakże nigdy nie będzie ono gorsze niż pierwotne. Otrzymany model potencjalnie usprawnionych zmian cykli świateł zostanie zaprezentowany użytkownikowi aplikacji za pomocą danych opisowych pokazujących, czy zaszła poprawa w przepływie pojazdów, a jeśli tak to jak znaczna. Udostępniony zostanie również podgląd przyspieszonej symulacji ruchu na skrzyżowaniu, opierającej się na danych z wcześniej dostarczonych nagrań z elementem losowości, tak aby można było zaobserwować potencjalną poprawę w formie zbliżonej do realistycznej, względem cykli świateł sprzed optymalizacji.

1.4. Porównanie z podobnymi rozwiązaniami

Przed rozpoczęciem prac nad aplikacją członkowie zespołu byli świadomi, że zagadnienie to wielokrotnie już wcześniej podejmowano, zarówno z perspektywy naukowej, jak i komercyjnej. Poniżej przedstawiono wybrane narzędzia, oferujące usprawnienie ruchu na skrzyżowaniach. Ich twórcy nie opisują stosowanych metod, a jedynie – w mniej lub bardziej dokładny sposób – funkcjonalności swoich produktów.

1.4.1. PTV Group

Niemiecka firma PTV Group oferuje kilka różnych produktów w zależności od przeznaczenia [8, 9]:

- PTV Epics — optymalizuje ruch na pojedynczych skrzyżowaniach. Model obserwuje warunki na skrzyżowaniu w czasie rzeczywistym i oblicza różne wersje sterowania przepływem, biorąc przy tym pod uwagę pieszych i samochody oraz transport publiczny. Parametry modelu tj. minimalna i maksymalna długość danego sygnału, fazy światła zielonego dla niekolizyjnych przejazdów czy wagi dla danego rodzaju uczestnika ruchu mogą być definiowane ręcznie, aby uzyskać preferowany efekt, np. priorytetyzację transportu publicznego.
- PTV Balance — optymalizuje i steruje ruchem sieci skrzyżowań. Ma za zadanie reagować na obecne natężenie ruchu i zwiększać efektywną przepustowość sieci dróg. Model, obliczając scenariusze cykli sygnalizacji, bierze pod uwagę wiele czynników m.in. czas oczekiwania na zielone światło, zagęszczenie ruchu na pojedynczych ulicach i bocznych drogach czy blokowanie się samochodów na kolejnych przejazdach. Następnie ewaluuje obliczone cykle za pomocą edytowalnego współczynnika, będącego średnią ważoną wszystkich ww. składowych.
- PTV Vissim — umożliwia kalibrację i testowanie powyższych produktów.

Produkty grupy są szeroko stosowane także przez inne firmy m.in. Gevas Software wykorzystującą PTV Balance [10] czy OneRoad korzystającą z PTV Vissim [11].



System PTV Epics jest w zamyśle bardzo zbliżony do naszego rozwiązania. Według zapewnień producenta ma oferować więcej dostosowywalnych parametrów i w przeciwieństwie do TFO ma działać w czasie rzeczywistym. Firma nie udostępnia żadnych konkretnych danych na jego temat, np. w jaki sposób system otrzymuje parametry czy dane o ruchu, zatem nie ma możliwości określenia dokładnych różnic. Główna idea obu aplikacji jest taka sama, czyli optymalizacja sygnalizacji na pojedynczym skrzyżowaniu, jednak bazując na dostępnej wiedzy TrafficFlowOptimizer ma być aplikacją łatwiejszą w obsłudze i stanowić rozwiązanie wymagające mniejszych zasobów sprzętowych.

1.4.2. Krakowski Obszarowy System Sterowania Ruchem (UTSC)

Według artykułu [12] dotyczącego UTSC z sierpnia 2013 roku „w Krakowie nie ma już sytuacji, w której jeden program sygnalizacji pracowałby jednakowo przez cały dzień. Programy sygnalizacji świetlnej zostały zróżnicowane i dostosowane do warunków ruchu o różnych porach doby i tygodni”. Informacje te potwierdza artykuł [13] z BIP, wskazujący, iż system ów to „modernizacja sygnalizacji świetlnej na kilkudziesięciu skrzyżowaniach, głównie w ciągu ul. Bronowickiej, al. Pokoju i Nowohuckiej, ale także na alejach Trzech Wieszczów. W rezultacie o ok. 10 proc. skrócono czas przejazdu komunikacji zbiorowej, a o 7 proc. szybciej jeżdżą samochody osobowe”.

UTSC składa się obecnie z dwóch współpracujących ze sobą komponentów: algorytmu MOTION firmy Siemens oraz systemu CROSSIG wykorzystującego PTV BALANCE dostarczanego przez Gevas Software. Ich głównymi zadaniami są: automatyczna optymalizacja pracy sygnalizacji dokonywana na podstawie pomiarów, koordynacja przejazdu na ciągach komunikacyjnych i przydzielanie priorytetu dla komunikacji zbiorowej oraz monitorowanie sytuacji drogowej w mieście i dostarczanie informacji o awariach czy usterkach. W UTSC można przełączać programy sterowania zarówno ręcznie, jak i automatycznie. Współpraca systemów jest możliwa dzięki obopólnemu wspieraniu standardu komunikacyjnego OCIT. Firma Gevas oprócz ww. systemu oferuje także inne produkty [10] mające na celu m.in. zarządzanie transportem publicznym czy optymalizację zielonej fali.

Optymalizacja odbywa się w następujący sposób. Z zaimplementowanych danych system w modelu sieci oblicza straty czasu dla wszystkich uczestników ruchu. Potem optymalizuje działanie programu do warunków ruchu i koordynując działania sygnalizacji, przesyła do ich sterowników wynik obliczeń co 5 lub 15 minut w zależności od tego, jak gwałtownie zmienia się natężenie ruchu. Urządzenia sterujące mają również wbudowane mechanizmy nadawania priorytetów. Służą one optymalizacji lokalnej, czyli tej wykonywanej dla sterowania sygnalizacją na danym skrzyżowaniu. Działa to według następującej zasady. Każdy pojazd ma przypisaną wagę, która jest przemnażana przez straty czasu i dzięki temu w funkcji otrzymywany jest efekt końcowy, jakim jest sumaryczna strata czasu na skrzyżowaniu. Wówczas sterownik stara się ją obniżyć, optymalizując program pracy sygnalizacji świetlnej.

System jest długoterminową inwestycją, która już w 2014 roku pochłonęła 47 mln zł i ma na celu optymalizację ruchu w całym mieście, przede wszystkim biorąc pod uwagę transport publiczny. W ścisłej z nim korelacji pracuje także System Nadzoru Ruchu Tramwajowego (TTSS).



Główną różnicą pomiędzy TrafficFlowOptimizer, a UTSC jest obszar, którym produkt się zajmuje. Podczas gdy krakowski system, obsługując pojedyncze skrzyżowania, dąży do maksymalizacji przepływu w całej sieci komunikacyjnej, celem TFO jest jedynie optymalizacja pojedynczego węzła komunikacyjnego. W przeciwieństwie do UTSC produkt nie jest tworzony z myślą o bezpośrednim połączeniu z siecią sygnalizacyjną oraz wymaga znacznie mniejszego budżetu.

1.4.3. Isarsoft

Firma Isarsoft [14] powstała w 2019 roku w Niemczech. Za pomocą narzędzia Perception, wykorzystującego sztuczną inteligencję, oferuje szereg rozwiązań związanych z mierzeniem ruchu na skrzyżowaniach, identyfikacją i unikaniem niebezpiecznych zdarzeń na drogach oraz optymalizacją przestrzeni parkingowych. Podczas gdy produkt jest przede wszystkim narzędziem analitycznym i nie ingeruje bezpośrednio w cykle sygnalizacji, pomaga zbierać dane konieczne do tego procesu w czasie rzeczywistym, takie jak zagęszczenie ruchu, typy przejeżdżających pojazdów czy ich prędkość. Ponieważ produkt opiera się na dostępie do kamer monitoringu, ważnym aspektem jest deklaracja firmy, dotycząca spełniania ogólnego rozporządzenia o ochronie danych, czyli RODO.



W niniejszej aplikacji model uczenia maszynowego analizujący obraz z nagrań jest jedynie środkiem koniecznym do uzyskania optymalizacji. W rozwiązaniu Isarsoftu stanowi on docelowy produkt. Firma przedstawia swój system jako wielofunkcyjny, jednak ma on głównie zastosowania pozwalające na obserwację i analizę danych, natomiast TrafficFlowOptimizer pozwala skupić się na wyselekcjonowanym problemie i zaproponować jego rozwiązanie. Obserwujemy zatem widoczną różnicę w koncepcji stosowania obu produktów do optymalizacji ruchu drogowego.

1.5. Główne funkcje systemu

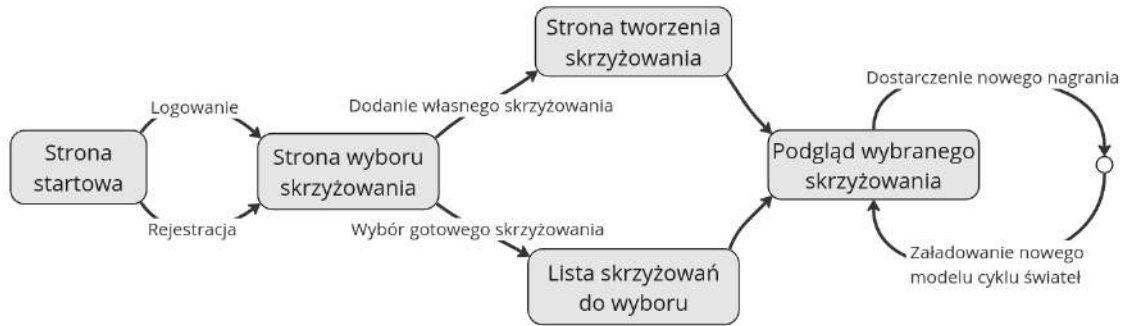
Użytkownik otrzymuje kilka możliwości wykorzystania naszego systemu (patrz rys. 1.5). Podstawowym i najważniejszym zastosowaniem jest wprowadzenie nowego skrzyżowania razem z nagraniami i wygenerowanie cyklu świateł. Po zakończeniu optymalizacji użytkownik ma możliwość porównania efektywności zoptymalizowanego skrzyżowania z rzeczywistym. Porównanie może być przedstawione w formie symulacji ruchu na skrzyżowaniu lub danych opisowych. Aplikacja zapewnia również dodatkową funkcję, jaką jest ponowne użycie już wprowadzonych skrzyżowań. Użytkownik może skorzystać z wprowadzonego już przez innych użytkowników skrzyżowania i wygenerować cykl świateł po wprowadzeniu nowych nagrań. W przypadku jeśli użytkownik nie chciałby, żeby wprowadzone przez niego dane były używane przez innych, może oznaczyć swoje skrzyżowania jako prywatne. W takiej sytuacji dane przypisane będą do konta i dostęp do nich będzie możliwy jedynie dla zalogowanego twórcy skrzyżowania oraz administratora.

Użytkownik może:

- skorzystać z wprowadzonego wcześniej skrzyżowania lub stworzyć własne
- skorzystać z wprowadzonych wcześniej nagrań pojazdów lub dodać własne
- wygenerować optymalny model cykli świateł dla wybranego przedziału czasowego
- zobaczyć porównanie podstawowego modelu wraz ze zoptymalizowanym modelem w formie symulacji lub danych opisowych
- usunąć stworzone wcześniej skrzyżowanie, w przypadku gdy jest jego autorem lub administratorem

Aplikacja zapewnia:

- uwierzytelnianie użytkowników i przechowywanie informacji o skrzyżowaniach i pojazdach w bazie danych,
- rozpoznawanie liczebności pojazdów na nagraniach za pomocą modelu uczenia maszynowego
- optymalizację przepływu pojazdów
- wizualizację zoptymalizowanych cykli świateł za pomocą uproszczonej symulacji rzeczywistego ruchu drogowego



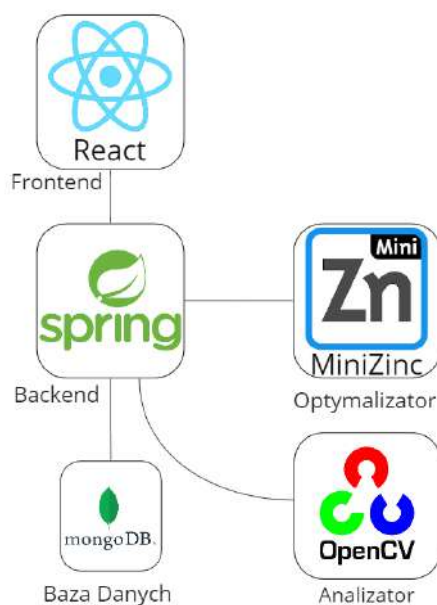
Rysunek 1.5: Schemat działania aplikacji

1.6. Główne założenia technologiczne

W dalszej części pracy pojęcie *Backend* (wielką literą) odnosić się będzie do komponentu aplikacji Backend, a nie ogółu serwisów działających na serwerze, zapewniających jej działanie. Analogicznie *Frontend*, *Analizator*, *Optymalizator* oraz *Baza Danych* (także wielką literą) stanowią referencję do komponentu aplikacji (patrz rys. 1.6).

Produkt to aplikacja webowa typu klient-serwer. Frontend zaimplementowany zostanie we frameworku React w języku TypeScript, Backend w Javie ze Springiem, Analizator będzie działać na podstawie narzędzia OpenCV, a Optymalizator będzie korzystać z MiniZinca. Te dwa ostatnie będą uruchomione poprzez Pythonowy framework FastAPI, który będzie wykorzystany do dostarczenia odpowiednio przygotowanych danych do komponentów.

W Optymalizatorze wykorzystany zostanie MiniZinc, korzystający z solvera CBC lub OR-Tools w zależności od tego, który z nich okaże się bardziej efektywny podczas testów oraz prostszy do integracji. Jako system zarządzania bazą danych użyty zostanie MongoDB. Do zliczania samochodów na skrzyżowaniu w Analizatorze użyjemy wytrenowanego wcześniej modelu wykorzystującego sieci neuronowe z użyciem biblioteki OpenCV, który będzie zliczał samochody przejeżdżające przez dane pasy.



Rysunek 1.6: Wykorzystywane technologie

1.7. Analiza ryzyka

Elementami, których implementacja niesie potencjalne ryzyko, są:

Skuteczny optymalizator

Może się okazać, że opracowany optymalizator nie będzie w pełni pokrywał funkcjonalności wszystkich typów świateł dostępnych na skrzyżowaniu — w szczególności chodzi o światła, które nie zapewniają bezkolizyjności lub stosowanie bus pasów. Kolejnym problemem może być zaimplementowanie w modelu świateł przy pasach pośrednich, nieznajdujących się bezpośrednio na wjeździe na skrzyżowanie. Ponadto istnieje ryzyko, że dla niektórych skrzyżowań problem okaże się na tyle złożony, że wyliczenie optymalnego modelu będzie zbyt kosztowne — w takich sytuacjach konieczne będzie uproszczenie modelu i rezygnacja z niektórych funkcjonalności. Innym podejściem może być porzucenie na nieoptymalnym modelu końcowym.

Model rozpoznawania samochodów i jakość dostarczonych nagrań

Nagrania z kamer będą różnić się m.in. kątem nagrywania, oświetleniem czy odległością, więc istnieje ryzyko, że konsystentne szczytywanie z nich informacji koniecznych do wykorzystywanego w pracy solvera będzie dla modelu wyzwaniem. Problematiczne może być przede wszystkim policzenie liczby samochodów, np. jeżeli model zgubi pojazd w momencie, kiedy powinno nastąpić zliczanie, to samochód nie zostanie policzony. Dojdzie wtedy do wygenerowania niepoprawnych danych, które mogą rzutować na wynik optymalizacji.

Autorzy pracy zakładają, że takie incydenty będą występować w częstotliwości błędu statystycznego i różnice przez to spowodowane okażą się nieistotne dla ostatecznego rezultatu optymalizacji. Jeżeli jednak taka sytuacja zostałaby zaobserwowana w większej, powodującej znaczące różnice skali, planowane jest przeprowadzenie odpowiedniego uzdatnienia zwracanych przez model wartości, np. poprzez ich uśrednianie lub skalowanie.

Przedstawienie wyników optymalizacji w formie symulacji

Problematiczne może być zamodelowanie ruchu na stworzonym skrzyżowaniu, po ukończeniu optymalizacji, jako że użytkownik w pełni samodzielnie określa jego dokładny schemat, na którym oparty będzie model. W szczególności gdy układ węzła drogowego będzie dosyć złożony, może pojawić się konieczność skorzystania ze znacznych uproszczeń. Niezależnie od poziomu skomplikowania wizualizowanego obiektu, wysoce prawdopodobna jest konieczność stworzenia nowej jego reprezentacji wyłącznie w ww. celu. Oczywiście tego następstwem jest wymagany większy nakład pracy deweloperskiej, szczególnie biorąc pod uwagę kłopoty w zachowaniu korelacji między modelem skrzyżowania użytym do optymalizacji oraz tym wykorzystywanym przy prezentacji wyników.

Wprowadzanie nowych skrzyżowań

W przypadku braku gotowego narzędzia pasującego do wymagań aplikacji zaimplementowanie własnego będzie problematyczne i kosztowne. Z pewnością zajmie zdecydowanie więcej czasu niż wkomponowanie gotowej biblioteki, a do tego prawdopodobnie nie będzie aż tak rozbudowane. Szczególnie kłopotliwe może również okazać się szczytywanie kolizji między przejazdami oraz zaznaczenie legalnych tras przejazdu pojazdów. Niezależnie

od sposobu implementacji tego fragmentu będzie on silnie związany z przyjętą reprezentacją skrzyżowania w bazie danych i wszystkie zmiany po którejkolwiek ze stron będą prawdopodobnie wymuszały zmiany po drugiej stronie.

Uwierzytelnianie użytkowników

Wstępne próby implementacji uwierzytelniania po stronie serwera, wykazały konieczność silnej integracji w tym zakresie między Backendem a Frontendem oraz uwydatniły kłopoty z przesyłaniem i przechowywaniem haseł w bezpiecznej postaci. Rozwiązanie tego problemu autorzy mają nadzieję znaleźć w gotowych bibliotekach oraz mechanizmach logowania i rejestracji, szeroko dostępnych zarówno po stronie klienta jak i serwera.

Pozorne powolne działanie aplikacji

Ze względu na potencjalnie złożone modele skrzyżowań, duży rozmiar danych wideo powstaje ryzyko długiego przesyłania danych między klientem a serwerem. Innym problemem może być długi czas działania optymalizatora. Wtedy użytkownik może odnieść wrażenie, że sama aplikacja jest powolna i nie działa płynnie, co będzie negatywnie rzutować na opinię o jakości produktu.

Rozwiązaniem tego problemu może być w niektórych przypadkach jawne przygotowanie klienta na konieczność oczekania przez pewien czas, co jest powszechnie stosowane w aplikacjach komercyjnych m.in. służących do edycji wideo. W innych sytuacjach autorzy dołożą starań, by maksymalnie usprawnić przesył danych i w miarę możliwości wykonywać go w tle, gdy użytkownik będzie już zajęty następnymi krokami korzystania z aplikacji, a w najgorszym razie autorzy pracy przewidują konieczność odwrócenia uwagi klienta poprzez wykorzystanie komponentów typu Call-to-Action, jak np. wyświetlenie reklamy lub prośba o ocenę produktu.

Rozdział 2

Zakres funkcjonalności

2.1. Użytkownicy systemu

Aplikacja umożliwia dostęp dwóm typom użytkowników — prywatnym oraz administratorowi poprzez uwierzytelnianie za pomocą logowania. Ponadto należy rozróżnić użytkowników konta prywatnego na specjalistów ruchu drogowego oraz użytkowników bez wiedzy specjalistycznej, lecz z dostępem do nagrań ruchu ulicznego.

2.1.1. Specjalista ruchu drogowego

Ten użytkownik z założenia zna zasady ruchu drogowego oraz fizyczny układ skrzyżowania. Jego zadaniem jest zainicjalizowanie skrzyżowania poprzez określenie pozycji wjazdów oraz wyjazdów, połączeń między nimi, a także świateł i możliwych potencjalnych kolizji. Jego odpowiedzialnością jest zamodelowanie skrzyżowania w taki sposób, aby poprawnie odzwierciedlało rzeczywisty układ. Aplikacja dopuszcza dużą dowolność parametrów i konfiguracji, ale równocześnie jest z tego powodu wrażliwa na błędy użytkownika.

2.1.2. Użytkownik z dostępem do nagrań ruchu drogowego

Ten użytkownik nie musi posiadać wiedzy na temat skrzyżowania — wystarczy że wybierze interesujące go skrzyżowanie z listy. Konieczne jest jednak, aby posiadał odpowiednio spreparowane nagranie ruchu ulicznego. Powinno być ono wykonane z odpowiedniej perspektywy oraz bez zakłóceń. Wówczas użytkownik może załadować nagranie dla wybranego skrzyżowania, co spowoduje przeliczenie nagrania na dane liczbowe opisujące gęstość ruchu dla odpowiadających wjazdów skrzyżowania. Po określeniu godzin, których dotyczy nagranie, może również zlecić optymalizację, która zwróci propozycję cykli świateł optymalizujących ruch dla nowo wyznaczonej charakterystyki przepływu.

2.1.3. Administrator

Użytkownik po zalogowaniu się na konto administratora oprócz możliwości wykonywania operacji dostępnych dla użytkownika prywatnego, może uzyskać dostęp do dowolnego skrzyżowania utworzonego przez któregośkolwiek z użytkowników. Ponadto ma on bezpośredni dostęp do informacji zawartych w bazie danych.

2.2. Zewnętrzne systemy współpracujące

Do stworzenia aplikacji wykorzystano następujące zewnętrzne systemy współpracujące.

2.2.1. Docker

Docker [15] jest narzędziem pozwalającym dostarczać programy w formie zwirtualizowanych kontenerów. W porównaniu do całej maszyny wirtualnej zapewnia lepszą przenośność, izolację, efektywność i skalowalność oprogramowania.



2.2.2. Maven

Maven [16] jest narzędziem automatyzującym budowę projektów na platformę Javy. Jest rozwijany przez The Apache Software Foundation. Pozwala w prosty i wygodny sposób z wykorzystaniem pliku XML konfigurować ustawienia projektu i definiować jego zależności.



2.2.3. Spring

Spring [17] jest frameworkiem do tworzenia aplikacji na platformę Javy. Ułatwia tworzenie aplikacji webowych, oferując moduły takie jak przykładowo:

- Spring Web
- Spring Security
- Spring Boot DevTools
- Spring Data



Pozwala uniknąć pisania tzw. kodu boilerplate i skupić się na implementacji logiki biznesowej. Charakteryzuje się Inversion of Control (IoC), co ułatwia konfigurację tudzież łączenie komponentów aplikacji. Wystarczy opisać obiekty i ich zależności, a Spring sam będzie zarządzał ich tworzeniem i odpowiednim powiązaniem.

2.2.4. Testcontainers

Testcontainers [18] jest frameworkiem służącym do testowania aplikacji. Na czas testów tworzy instancje serwisów na podstawie podanych obrazów. Pozwala to, przykładowo przetestować napisane bazodanowe serwisy podając obraz i wersję bazy danych używanej w testowanej aplikacji. Dzięki temu można sprawdzić działanie kodu na środowisku przypominającym produkcyjne lepiej niż za pomocą mock testów.



2.2.5. Mockito

Mockito [19] jest frameworkiem do testów. Umożliwia tworzenie tzw. zmockowanych Javowych obiektów. Są to atrapy prawdziwych obiektów, dla których definiujemy konkretne działanie, aby sprawdzić poprawność wybranego komponentu aplikacji.



2.2.6. MongoDB i GridFS

MongoDB [20] jest nierelacyjnym systemem zarządzania bazą danych. Przechowuje obiekty w formie dokumentów o strukturze podobnej do formatu JSON. Charakteryzuje się wygodą w przechowywaniu słabo ustrukturyzowanych danych oraz efektywnością horyzontalnej skalowalności względem relacyjnych baz danych. W aplikacji wykorzystany jest także GridFS umożliwiający przechowywanie większych plików w Mongo, w szczególności nagrań.



2.2.7. FastAPI

FastAPI [21] to najczęściej wykorzystywany Pythonowy framework do tworzenia API w architekturze REST. Jego popularność wynika z wygody, prostoty, szybkości oraz intuicyjności użytkowania.



2.2.8. OpenCV

OpenCV [22] (Open Source Computer Vision Library) jest biblioteką oferującą dużą liczbę funkcjonalności związanych z analizą obrazu w czasie rzeczywistym. Posiada interfejsy w wielu popularnie wykorzystywanych językach programowania, takich jak Python, Java oraz C++.



2.2.9. CBC

CBC [23] (COIN-OR Branch and Cut) to solver open source, stworzony w ramach projektu fundacji Computational Infrastructure for Operations Research [24], zaimplementowany w języku C++. Opiera się on na MIP (Mixed Integer Programming), wykorzystując technikę Branch and Cut. Został wykorzystany jako zewnętrzny podwykonawca optymalizacji.



2.2.10. Gecode

Gecode [25] to również solver open source oparty o język C++, jednak znacznie mniej wydajny niż CBC. Jest on często wykorzystywany jako zabezpieczenie, w przypadku niemożliwości skorzystania z bardziej zaawansowanych solverów.

2.2.11. Google Maps

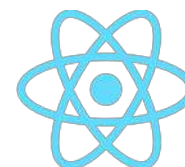
Google Maps to internetowa platforma umożliwiająca oglądanie satelitarnych zdjęć ulic. W aplikacji jest wykorzystywana w postaci Google Maps Embed [26] w celu pozyskania zdjęcia skrzyżowania, które jest wymagane na etapie tworzenia modelu dopasowanego do potrzeb użytkownika.



2.2.12. React

React [27] jest frameworkiem oferującym alternatywne podejście do tworzenia graficznych interfejsów użytkownika w przeglądarce oraz aplikacjach mobilnych (wersja react-native). Do jego głównych zalet zalicza się:

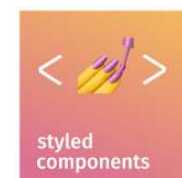
- kompaktowość tworzonego kodu (połączenie HTML'a oraz kodu Javascriptu przy pomocy rozszerzenia JSX)
- możliwość podziału interfejsu na pomniejsze części, tzw. komponenty, które można wykorzystywać wielokrotnie
- wbudowane funkcje pozwalające panować nad stanem interfejsu — znane jako hooki
- możliwość skorzystania z wielu zewnętrznych bibliotek, ułatwiających m.in. stylowanie interfejsu czy wykonywanie zapytań do serwera



2.2.13. Styled-components i Material UI

Są to dwie biblioteki wspierające stylowanie stron internetowych lub aplikacji mobilnych:

- Ideą styled-components [28] jest tworzenie styli od podstaw. Są one określane w języku CSS dla pojedynczych tagów języka HTML. Tak tworzone stałe możemy wstawiać do JSX'a zamiast ww. tagu. Ułatwia to organizację kodu oraz wielokrotne wykorzystywanie tych samych styli. Oprócz Reacta biblioteka wspiera także Vue.js.
- MaterialUI [29] natomiast przeznaczona jest wyłącznie dla Reacta. Oferuje gotowe komponenty, np. tooltip czy alert, które można od razu wykorzystać w projekcie, bez konieczności określania styli samemu.



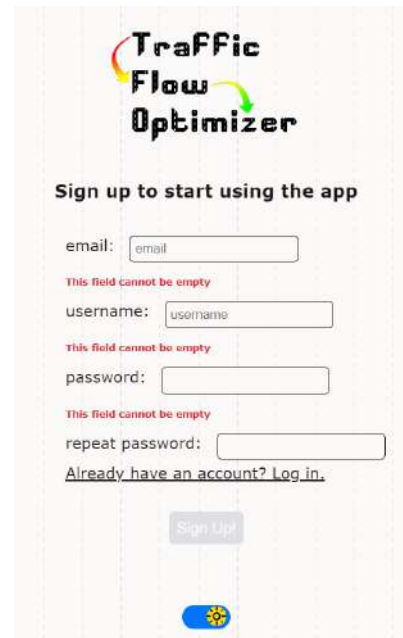
2.3. Wymagania

2.3.1. Wymagania funkcjonalne

Aplikacja TFO spełnia następujące wymagania funkcjonalne

Rejestracja

Aby korzystać z aplikacji, użytkownik musi posiadać konto. Umożliwia mu to mechanizm rejestracji. Musi wybrać między innymi unikalną nazwę użytkownika, adres email i hasło. Podane wartości są walidowane, aby wykryć i uniknąć prostych błędów.



The screenshot shows the registration interface for the 'Traffic Flow Optimizer' app. At the top is the app's logo. Below it, the text 'Sign up to start using the app' is displayed. The form includes four input fields: 'email', 'username', 'password', and 'repeat password'. Each field has a placeholder text and a red error message 'This field cannot be empty' below it. At the bottom of the form is a 'Sign Up!' button and a link that says 'Already have an account? Log in.'.

Logowanie



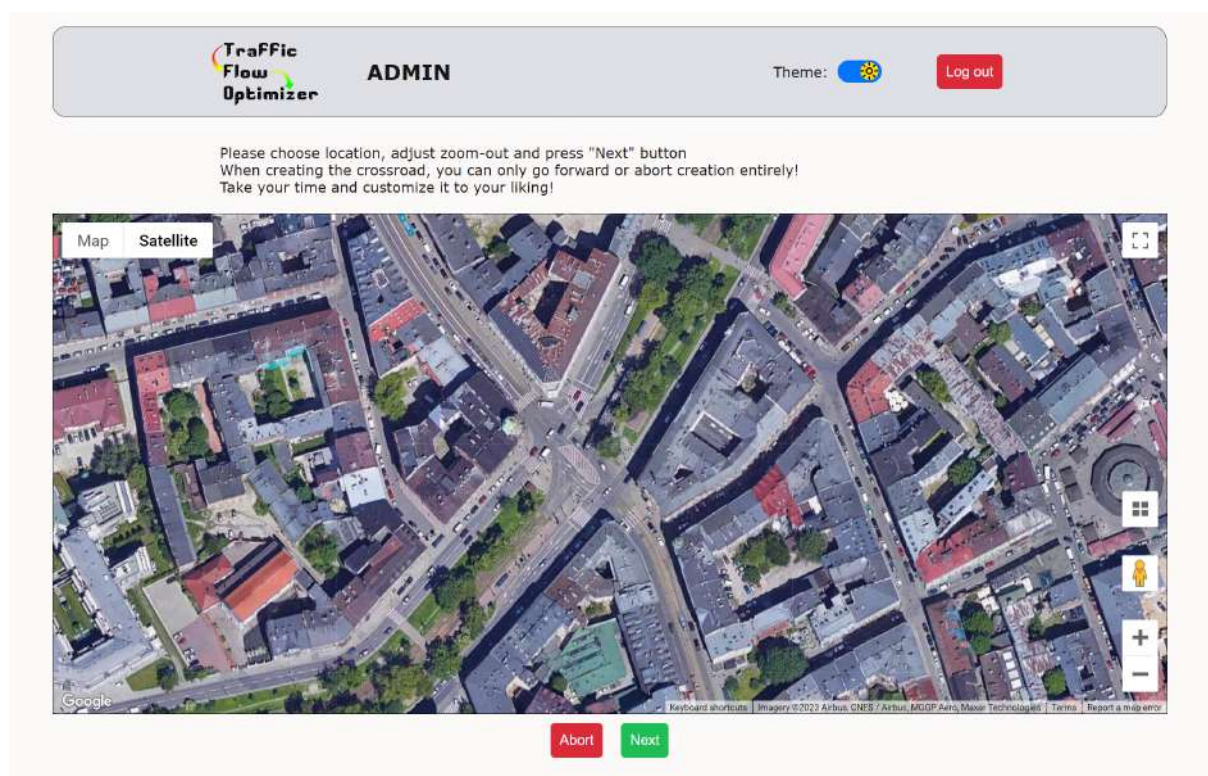
The screenshot shows the login interface for the 'Traffic Flow Optimizer' app. At the top is the app's logo. Below it, the text 'Log in to proceed' is displayed. The form includes two input fields: 'username' and 'password'. Each field has a placeholder text and a red error message 'This field cannot be empty' below it. At the bottom of the form is a 'Log In!' button and a link that says 'Don't have an account? Register now.'.

Użytkownik może zalogować się na swoje konto, podając nazwę użytkownika oraz hasło. Logowanie zapewnia mu dostęp do funkcjonalności aplikacji oraz zapisanych przez siebie obiektów.

Tworzenie skrzyżowania w intuicyjny sposób

Użytkownik może zainicjalizować dowolne skrzyżowanie za pomocą autorskiego narzędzia. W trakcie tego procesu otrzymuje krótkie instrukcje, opisujące co należy zrobić na danym etapie. Dodawanie kolejnych elementów do skrzyżowania odbywa się na wyciętym zdjęciu z obrazu satelitarne, pomagającego w wyobrażeniu sobie jego układu (patrz rys. 2.1).

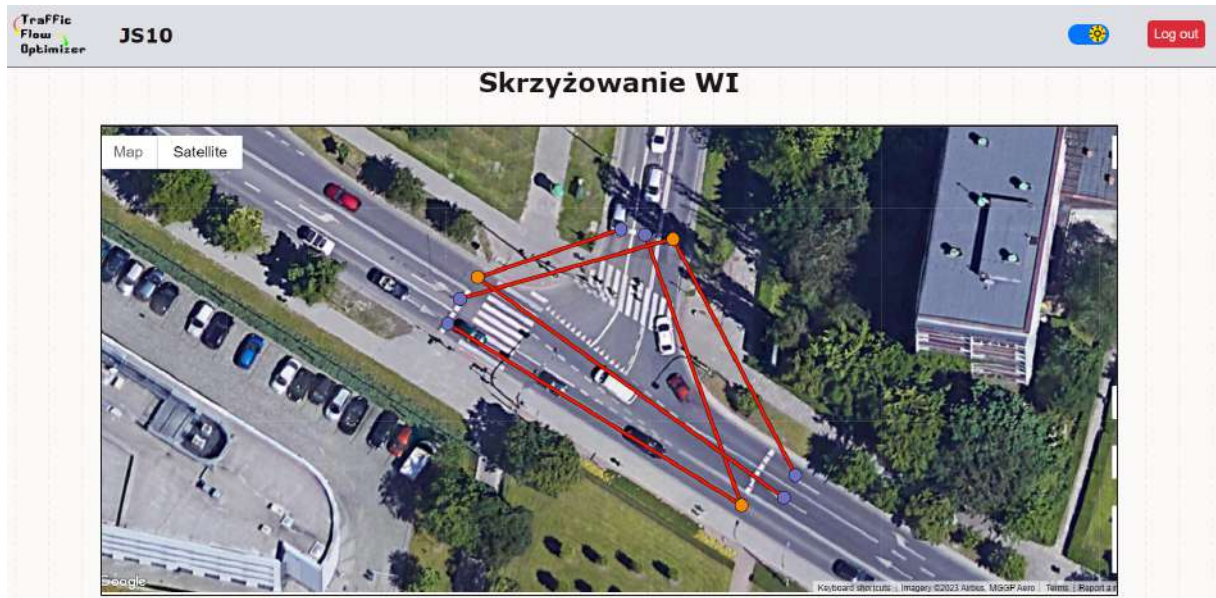
Narzędzie daje dużą dowolność, aby można było skutecznie odwzorować różne skrzyżowania. W szczególności umożliwia określenie miejsc, gdzie znajdują się wjazdy oraz wyjazdy i zadeklarowanie połączeń między nimi. Określa również jakiego typu światła znajdują się przy którym wjeździe, a także które z nich kolidują ze sobą. Proces jest stosunkowo złożony, jednak dzięki temu użytkownik ma możliwość odwzorowania dowolnego, wybranego przez siebie skrzyżowania.



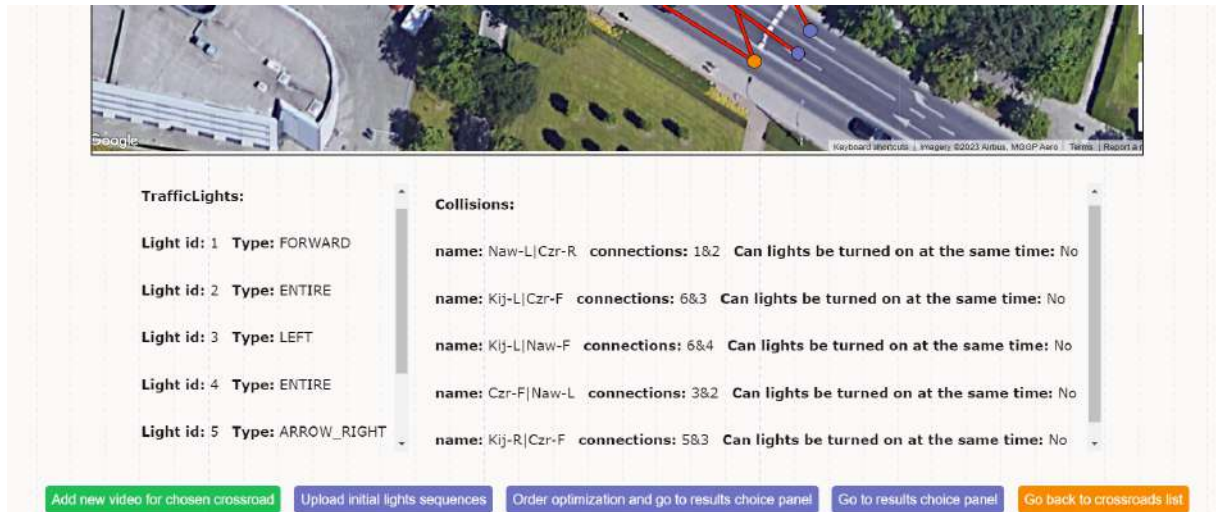
Rysunek 2.1: Jeden z ekranów tworzenia skrzyżowania

Możliwość podglądu istniejącego skrzyżowania

Użytkownik ma możliwość podglądu dowolnego, stworzonego już skrzyżowania, jeśli tylko pozwalają na to ustawienia prywatności. Zostają mu przedstawione wszystkie jego komponenty w postaci graficznej oraz w formie listy (patrz odpowiednio rys. 2.2 i 2.3).



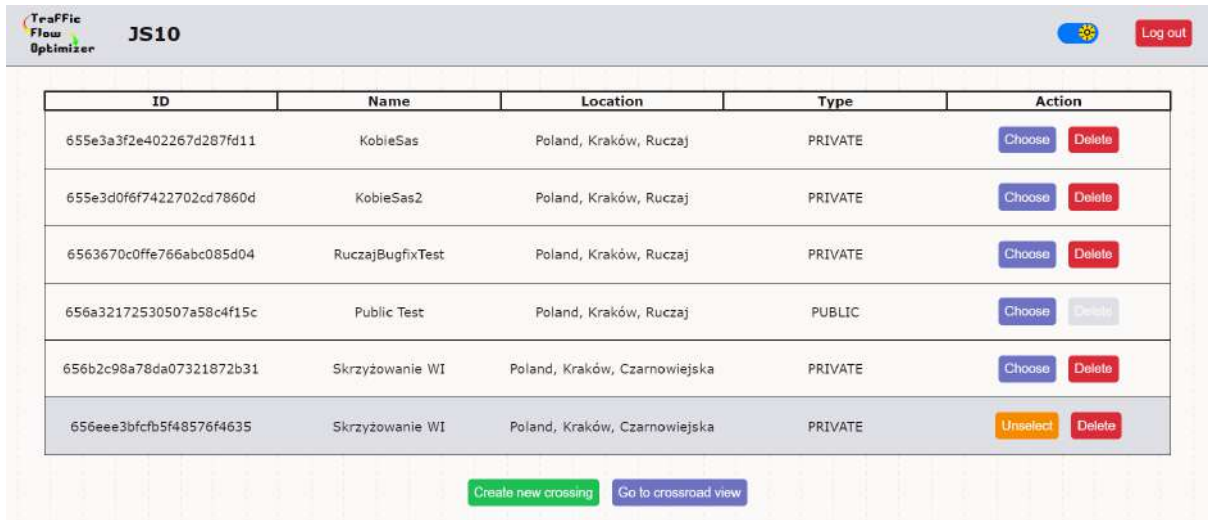
Rysunek 2.2: Ekran podglądu skrzyżowania cz. 1.



Rysunek 2.3: Ekran podglądu skrzyżowania cz. 2.

Przechowywanie skrzyżowań w bazie danych

Użytkownik ma możliwość przechowywania stworzonych przez siebie skrzyżowań w bazie danych, co umożliwia mu dostęp do nich z innego urządzenia. Ma także uprawnienia do usuwania i podglądu zapisanych przez siebie modeli (patrz rys. 2.4).



ID	Name	Location	Type	Action
655e3a3f2e402267d287fd11	KobieSas	Poland, Kraków, Ruczaj	PRIVATE	Choose Delete
655e3d0f6f7422702cd7860d	KobieSas2	Poland, Kraków, Ruczaj	PRIVATE	Choose Delete
6563670c0ffe766abc085d04	RuczajBugfixTest	Poland, Kraków, Ruczaj	PRIVATE	Choose Delete
656a32172530507a58c4f15c	Public Test	Poland, Kraków, Ruczaj	PUBLIC	Choose Delete
656b2c98a78da07321872b31	Skrzyżowanie WI	Poland, Kraków, Czarnowiejska	PRIVATE	Choose Delete
656eee3bfcfb5f48576f4635	Skrzyżowanie WI	Poland, Kraków, Czarnowiejska	PRIVATE	Unselect Delete

[Create new crossing](#) [Go to crossroad view](#)

Rysunek 2.4: Ekran listy skrzyżowań użytkownika

Dodawanie nagrania

Użytkownik może dodać nagranie po sprecyzowaniu których godzin ono dotyczy (patrz rys. 2.5). Musi on jednak pozyskać nagranie samemu, gdyż aplikacja nie pomaga w ich uzyskiwaniu.

Add videos for crossroad: Skrzyżowanie WI

Remember that the quality of analysis depends on the quality of a video.
It's on you to provide the recording with the correct angle and right illumination.
The best angle is from the side and slightly above the cars.
Thank you for your consideration!

Drag and drop your file here
or
Upload a file

Send video to server Open Detection Rectangles Creator

Select hour when the video was recorded ▼

Select day when the video was recorded ▼

Video real time:*

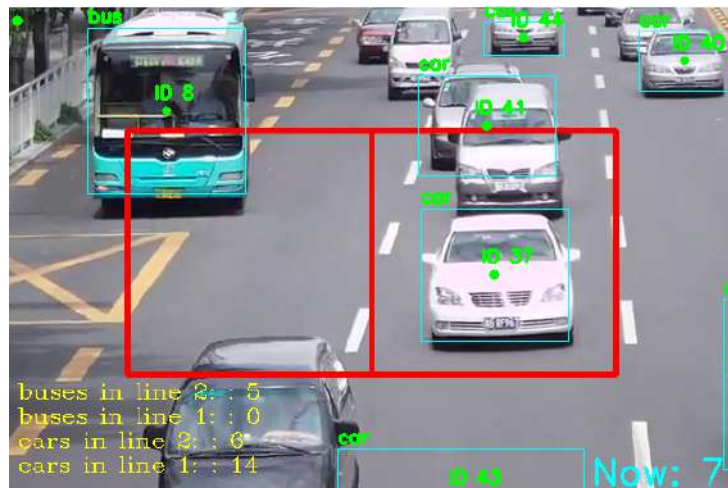
How much time is visualized on video in minutes ex: if video is 5 mins long, but it displays events in x2 speed, then the inserted value should be 10

Back to crossing view

Rysunek 2.5: Ekran dodawania nagrania

Zlecenie analizy nagrania

Po zdefiniowaniu obszarów, odpowiadających pasom ruchu drogowego, użytkownik może zlecić policzenie pojazdów przejeżdżających przez nie na nagraniu. Wynik dla każdego z obszarów zawiera informację o liczbie samochodów osobowych i autobusów (patrz rys. 2.6).



Rysunek 2.6: Wizualizacja efektów wykrywania pojazdów na nagraniu.

Zlecenie optymalizacji

Użytkownik może zlecić wykonanie optymalizacji, wybierając odpowiednie skrzyżowanie oraz godzinę nagrania. Określa także, ile czasu chce poświęcić na wyliczanie optymalizacji, potencjalnie uzyskując rezultat szybciej kosztem jego efektywności (patrz rys. 2.7). Optymalizacja zostaje zlecona na podstawie danych pozyskanych wcześniej z nagrań.

Crossroad: Skrzyżowanie WI

Optimization Time:

no time limit ▼

Hour and day to optimize:

Select hour when the video was recorded ▼

Select day when the video was recorded ▼

Close

Start optimization

Rysunek 2.7: Ekran zlecenia optymalizacji

Podgląd wyników optymalizacji

Użytkownik może zobaczyć rezultat optymalizacji w dwóch formach — symulacji lub spisu kolejnych cykli świateł (patrz odpowiednio rys. 2.8 i 2.9). Symulacja pokazuje, zmieniające się w kolejnych jednostkach czasu, kolory świateł oraz ilości samochodów oczekujących na przejazd na poszczególnych pasach, a co za tym idzie, pozwala oszacować czy faktycznie zmniejszono poziom zatorów.



Rysunek 2.8: Ekran wyników w postaci symulacji

Natomiast spis umożliwia porównanie obecnych cykli świateł względem tych sprzed optymalizacji.



Rysunek 2.9: Fragment ekranu wyników w postaci danych opisowych

2.3.2. Wymagania niefunkcjonalne

Aplikacja TFO zaspokaja następujące wymagania niefunkcjonalne

Rejestracja

- Konto jest gotowe do użycia od razu po zarejestrowaniu
- Dane użytkownika są walidowane po stronie klienta i serwera
- Gwarantowanie unikatowości nazwy użytkownika oraz jego adresu email
- Po rejestracji wybrane hasło zapisywane jest w bazie danych w zaszyfrowanej postaci

Logowanie

- Użytkownik może być zalogowany na konto na kilku różnych urządzeniach na raz
- Dane logowania są walidowane zarówno po stronie klienta jak i serwera
- Przy kolejnych interakcjach z serwerem zalogowany użytkownik jest identyfikowany za pomocą tokenu JWT, który otrzymuje po poprawnym logowaniu

Tworzenie skrzyżowania w intuicyjny sposób

- Tworzenie skrzyżowania zajmuje 10–30 min w zależności od jego złożoności
- Na każdym etapie tworzenia modelu użytkownik ma dostęp do krótkiej instrukcji obsługi
- Tworzenie skrzyżowania jest na tyle intuicyjne, że nie wymaga samouczka
- Użytkownik w momencie tworzenia danego typu elementu może dodawać, usuwać oraz edytować go, aż do uzyskania zamierzonego efektu
- Użytkownik może w dowolnym momencie porzucić tworzenie modelu
- Użytkownik może stworzyć model skrzyżowania, które realnie nie istnieje. Aplikacja nie wymaga jego autentyczności
- Model zawiera wszystkie informacje niezbędne dla optymalizatora

Możliwość podglądu istniejącego skrzyżowania

- Skrzyżowanie na podglądzie wygląda tak samo jak podczas tworzenia przez użytkownika
- Wszystkie dane wprowadzone podczas modelowania są dostępne na podglądzie
- Z podglądu można przejść do zlecenia optymalizacji lub dodawania nagrania dla tego samego skrzyżowania

Przechowywanie skrzyżowań w bazie danych

- Użytkownik ma dostęp do listy stworzonych przez siebie skrzyżowań w dedykowanym widoku
- Użytkownik ma dostęp do wszystkich publicznych modeli

Dodawanie własnego nagrania

- Nagranie jest zapisywane w takiej samej jakości, jaką ma oryginał
- Użytkownik musi określić, której godziny oraz dnia, a także którego wjazdu na skrzyżowanie dotyczy nagranie
- Użytkownik może dodać nagranie dla tego samego wjazdu i godziny oraz dnia wiele razy, ale nowo dodany film zawsze nadpisuje poprzedni

Zlecenie analizy nagrania

- Analiza nagrania jest w najgorszym przypadku kilkukrotnie dłuższa niż nagranie
- Nie jest konieczne, aby nagranie trwało określoną ilość czasu, ponieważ wyniki zostają uśrednione do liczby aut na minutę
- Użytkownik musi wyznaczyć obszary przejazdów na pierwszej klatce nagrania w dedykowanym interfejsie na stronie aplikacji

Zlecenie optymalizacji

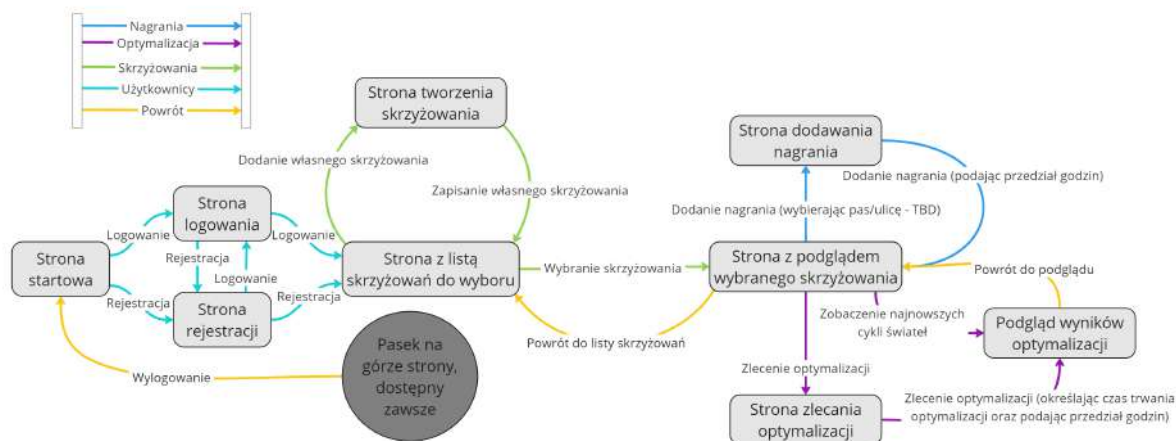
- Użytkownik musi wybrać czas działania optymalizatora i godzinę, dla której optymalizacja się odbywa
- Jeśli dla danej godziny nie dodano nagrania użytkownik nie może zlecić dokonania optymalizacji dla wybranej godziny
- Użytkownik podczas oczekiwania na rezultaty jest zajmowany informatycznymi żartami w postaci memów

Podgląd wyników optymalizacji

- Wyniki można wyświetlić w formie symulacji lub danych opisowych
- Symulacja wyświetla liczby samochodów oczekujących na poszczególnych pasach
- Symulacja pozwala użytkownikowi znaleźć część skrzyżowania zmniejszającą płynność ruchu w łatwy do zauważenia sposób
- Symulacja przedstawia informacje o rodzaju świateł, za pomocą umownych symboli
- Dane opisowe prezentują dla każdego przejazdu sekwencję kolorów sygnalizatorów na przestrzeni minuty oraz stosunki samochodów, które z danego wjazdu skorzystały do tych, które do niego dotarły, dla aktualnych cykli świateł oraz tych sprzed optymalizacji

2.4. Scenariusze użytkowania

Przepływ TFO z punktu widzenia użytkownika prezentuje się następująco (patrz rys. 2.10). Poszczególne scenariusze użytkowania opisano bardziej szczegółowo w poniższych punktach.



Rysunek 2.10: Przepływ aplikacji

Większość z poniższych scenariuszy zaczyna się od ekranu listy skrzyżowań (patrz rys. 2.11), gdzie można albo stworzyć skrzyżowanie, albo zobaczyć podgląd już zaznaczonego.

Traffic Flow Optimizer JS10 Log out				
ID	Name	Location	Type	Action
655e3a3f2e402267d287fd11	KobieSas	Poland, Kraków, Ruczaj	PRIVATE	<button>Choose</button> <button>Delete</button>
655e3d0f6f7422702cd7860d	KobieSas2	Poland, Kraków, Ruczaj	PRIVATE	<button>Choose</button> <button>Delete</button>
6563670c0ffe766abc085d04	RuczajBugfixTest	Poland, Kraków, Ruczaj	PRIVATE	<button>Choose</button> <button>Delete</button>
656a32172530507a58c4f15c	Public Test	Poland, Kraków, Ruczaj	PUBLIC	<button>Choose</button> <button>Delete</button>
656b2c98a78da07321872b31	Skrzyżowanie WI	Poland, Kraków, Czarnowiejska	PRIVATE	<button>Choose</button> <button>Delete</button>
656eee3bfcfb5f48576f4635	Skrzyżowanie WI	Poland, Kraków, Czarnowiejska	PRIVATE	<button>Unselect</button> <button>Delete</button>

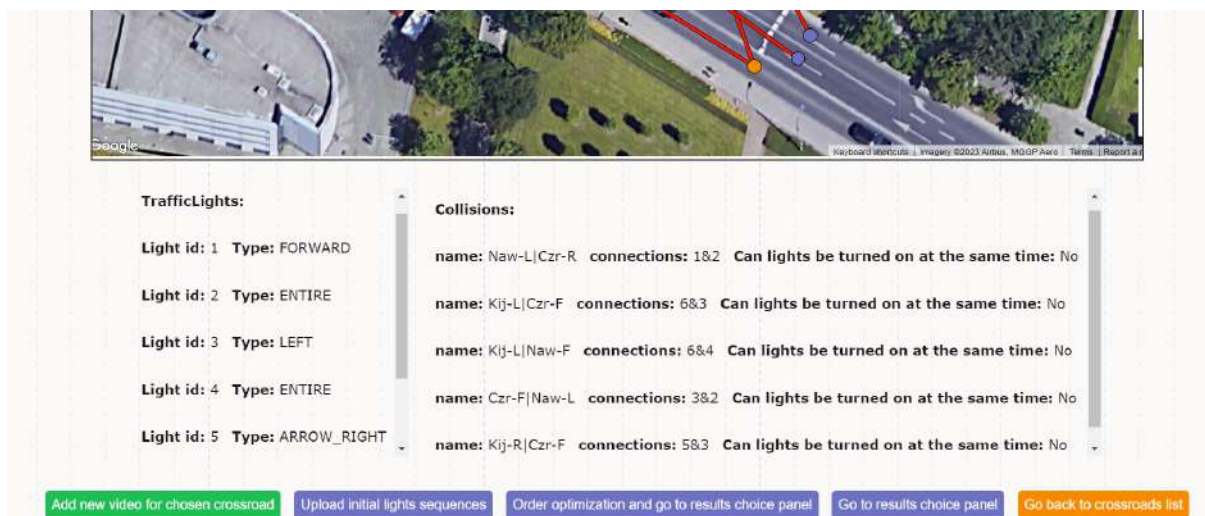
Create new crossing Go to crossroad view

Rysunek 2.11: Lista skrzyżowań

Po wybraniu podglądu skrzyżowania można wybrać jedną z kilku możliwych akcji (patrz rys. 2.12):

1. Dodanie nagrania
2. Dodanie pierwotnej sekwencji sygnalizacji świetlnej
3. Zlecenie optymalizacji oraz podgląd jej wyników
4. Podgląd najnowszych wyników bez konieczności zlecenia optymalizacji

5. Powrót do listy skrzyżowań



Rysunek 2.12: Podgląd skrzyżowania


2.4.1. Tworzenie skrzyżowania

W tym scenariuszu użytkownik przechodzi przez osiem etapów na przestrzeni siedmiu widoków. Każdy z nich jest kolejnym krokiem tworzenia modelu i dodaje do niego nowe elementy. Każdy widok składa się z krótkiej instrukcji postępowania oraz podglądu obecnego stanu etapu, czyli ile nowych elementów zostało dodanych i jakie mają cechy. Dane określające nowo powstały element użytkownik podaje w formularzach znajdujących się albo bezpośrednio w widoku, albo w pojawiającym się modalu. Jedynym widokiem dwufazowym jest ten dotyczący świateł, gdzie najpierw są one tworzone, a następnie przypisywane do konkretnych przejazdów. Po przejściu do kolejnego etapu użytkownik nie ma możliwości cofnięcia się, ale może w dowolnym momencie porzucić tworzenie modelu poprzez naciśnięcie przycisku „Abort”.

1. Wybór tła skrzyżowania na podstawie jego lokalizacji w widoku z Google Maps



2. Wprowadzenie ogólnych informacji dotyczących obiektu



Basic information:

Name: Country: City: District:

Type: PUBLIC ☒ PRIVATE ☐

Inputs confirmed!

3. Stworzenie punktów reprezentujących wjazdy i wyjazdy

Entrances & Exits
Please follow these steps:
1. Click on the map in place where you want your entrance/exit/intermediate point to be
2. Fill-in the input fields in the creator and save the point
3. Repeat steps 1-2 for all entrances, exits and inters you need



Map Satellite

id: 3
type: EXIT
street: Zbyszewski
capacity: infinity

Fill in the data
In capacity type a positive integer or "infinity". Number is only relevant in case of intermediate points.

Type:

☒ ENTRANCE
☐ EXIT
☐ INTERMEDIATE

ID: 1

Name:

Capacity:

Inputs confirmed!

4. Stworzenie przejazdów

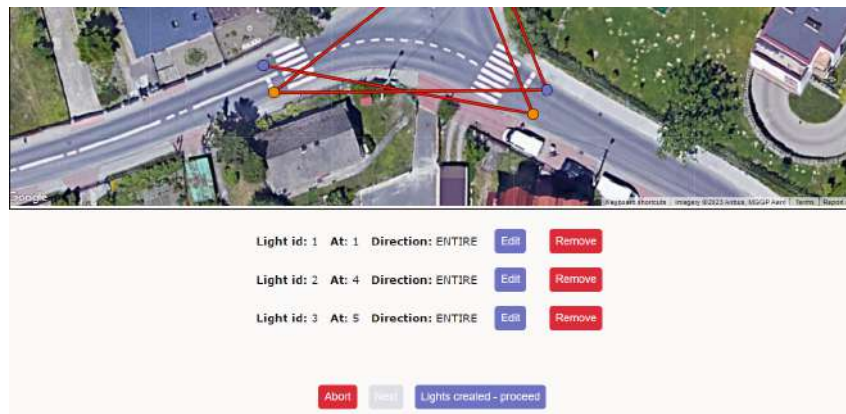


Source: 5
Target: 2

Name:

New connection added!

5. Stworzenie świateł



Traffic Light Creator

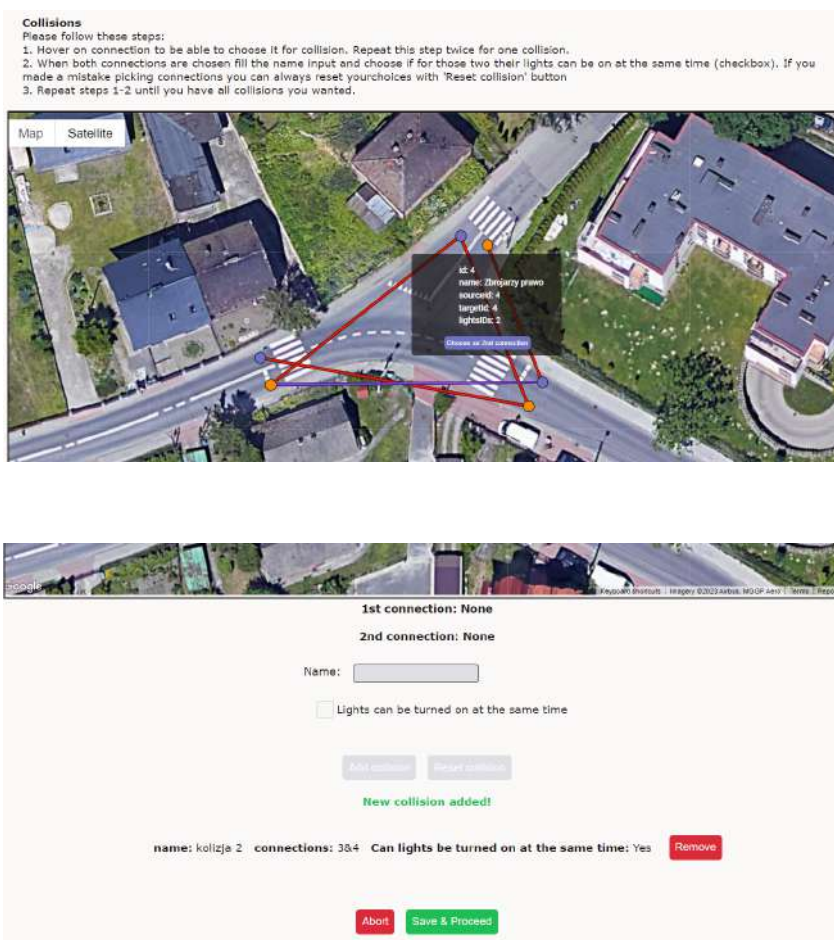
ID: 1

Confirm your choices

6. Przypisanie świateł do przejazdów



7. Stworzenie kolizji i zapisywanie



2.4.2. Dodawanie pierwotnej sekwencji sygnalizacji świetlnej


2.4.3. Dodawanie nagrania

W tym scenariuszu użytkownik korzysta z jednego widoku głównego, w którym przesyła nagranie, określając jego miejsce, godzinę i dzień oraz jednego widoku pomocniczego, w którym na pierwszej klatce nagrania wyznacza obszary pasów zawartych na nagraniu. Użytkownik może w każdej chwili przerwać czynność dodawania nagrania. Konsekwencją ukończenia pełnego scenariusza jest przesłanie wideo do analizy co skutkuje automatycznym przetworzeniem nagrania przez Backend i Analizator.

1. Dodanie nagrania z własnego urządzenia oraz określenie, kiedy zostało wykonane

Add videos for crossroad: Skrzyżowanie WI

Remember that the quality of analysis depends on the quality of a video.
It's on you to provide the recording with the correct angle and right illumination.
The best angle is from the side and slightly above the cars.
Thank you for your consideration!



Drag and drop your file here
or

Upload a file

Send video to server

Open Detection Rectangles Creator

Select hour when the video was recorded ▼

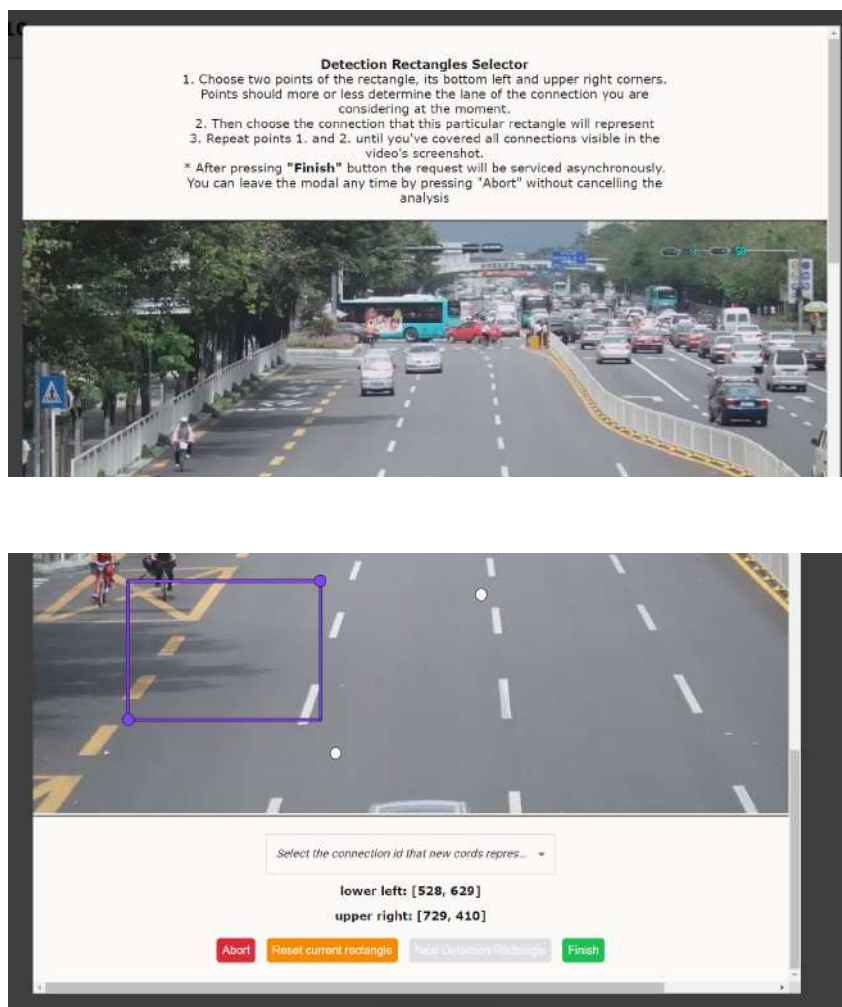
Select day when the video was recorded ▼

Video real time:*

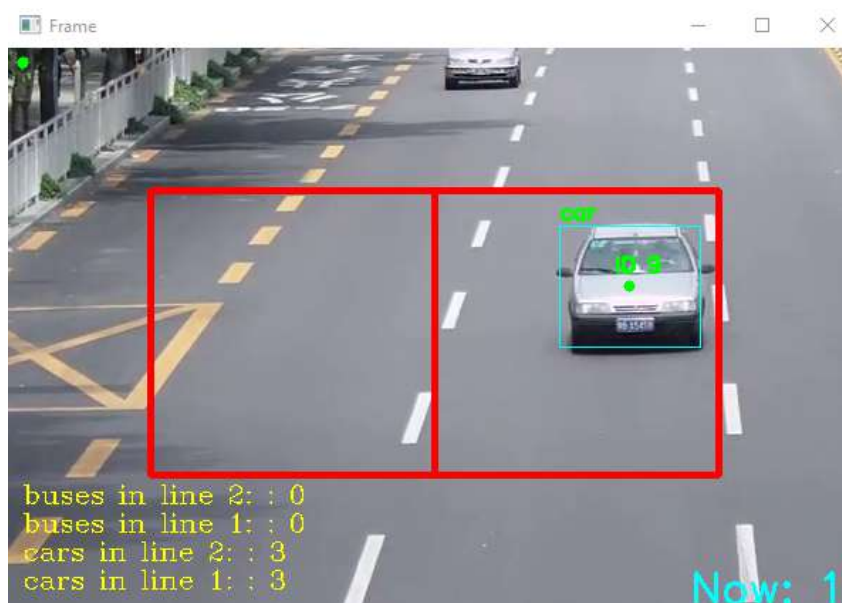
How much time is visualized on video in minutes ex: if video is 5 mins long, but it displays events in x2 speed, then the inserted value should be 10

Back to crossing view

2. Wyznaczenie obszarów odpowiednich przejazdów na klatce z nagrania



3. Zlecenie analizy przekazanego wcześniej nagrania



2.4.4. Zlecenie optymalizacji

1. Określenie czasu trwania optymalizacji i zlecenie jej rozpoczęcia

Crossroad: Skrzyżowanie WI

Optimization Time:

no time limit ▼

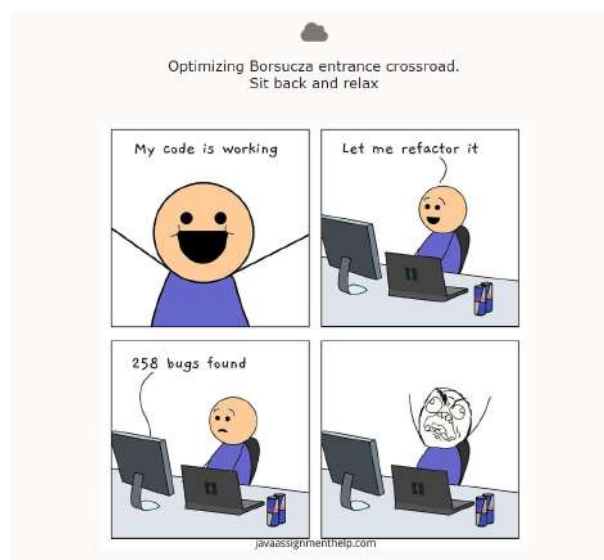
Hour and day to optimize:

Select hour when the video was recorded ▼

Select day when the video was recorded ▼

Close Start optimization

2. Oczekiwanie na wykonanie optymalizacji

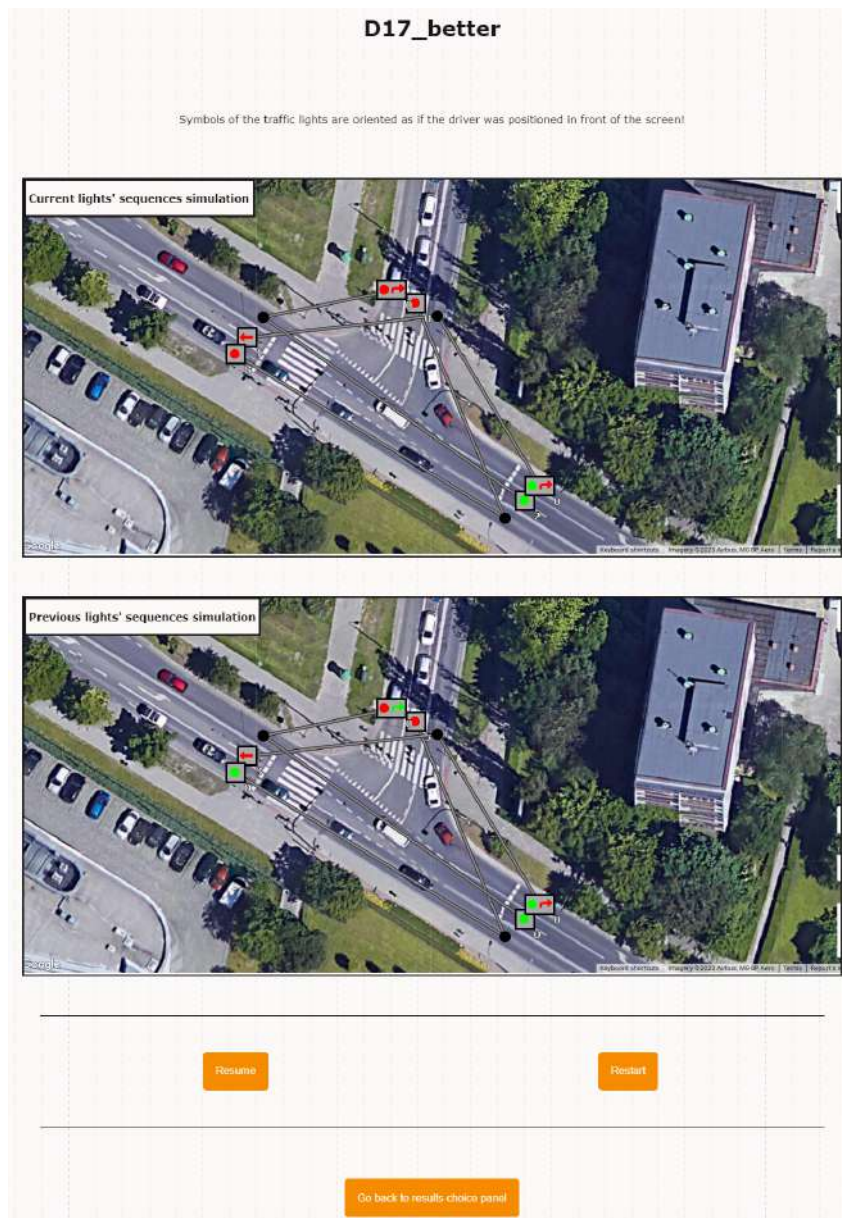
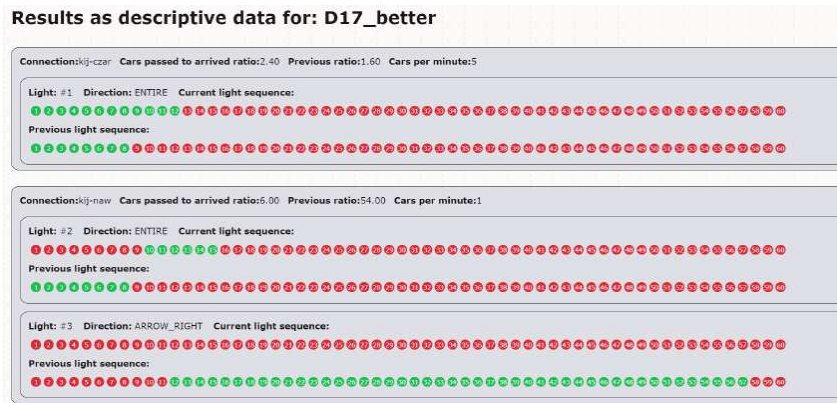


3. Wybór, w jakiej formie wyniki powinny zostać zaprezentowane

See results for optimization of Skrzyżowanie WI

Now choose your course of action:

See results as descriptive data See results as a simulation Go back to crossroad view



Ekran przedstawia dwie symulacje sekwencji świateł: poprzedniej oraz najnowszej, uzyskanej w zleconej optymalizacji. Ma to na celu ułatwienie ewaluacji jej rezultatów. Obie symulacje są sterowane jednym panelem. Pierwszy obraz prezentuje ekran symulacji przed jej uruchomieniem, a drugi już podczas jej trwania.

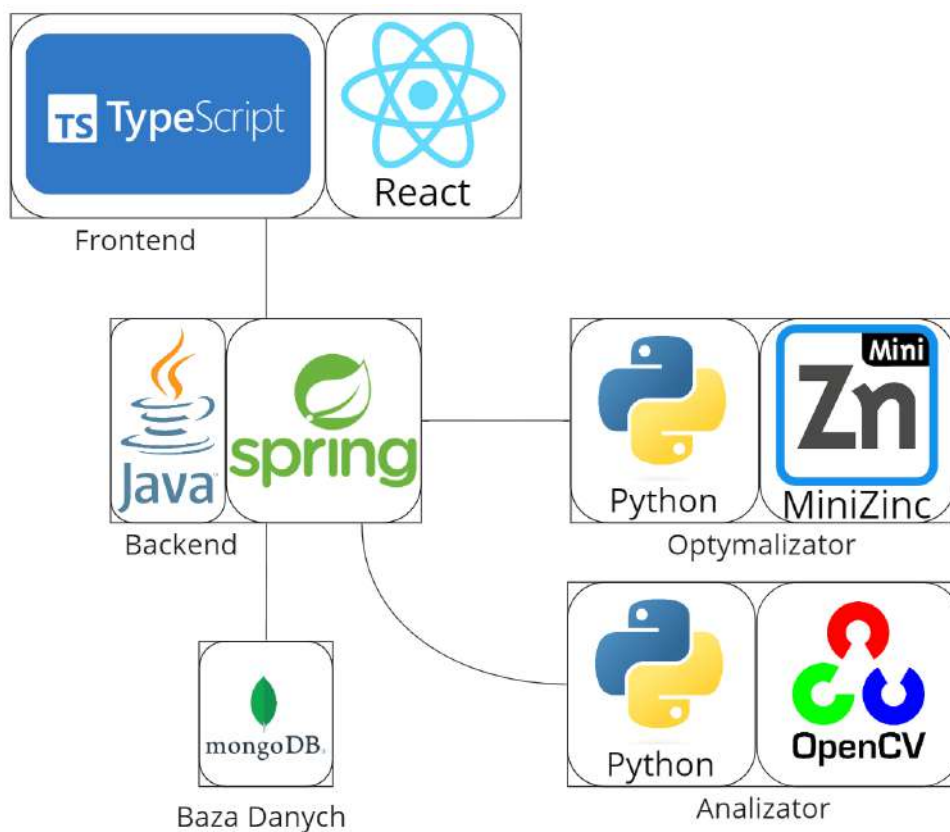
2.4.5. Podgląd najnowszej optymalizacji

Proces przebiega analogicznie jak w powyższym przypadku, jednak na etapie podglądu skrzyżowania należy wybrać „Go to results choice panel” zamiast „Order optimization and go to results choice panel”. Spowoduje to pominięcie punktów 1. oraz 2. i bezpośrednie przejście do ekranu wyboru sposobu prezentacji wyników optymalizacji.

Rozdział 3

Wybrane aspekty realizacji

Poniższa grafika przedstawia ogólny podgląd na architekturę projektu.



Rysunek 3.1: Architektura projektu

Można wyróżnić pięć komponentów (patrz rys. 3.1):

- Frontend
- Backend
- Baza Danych
- Optymalizator
- Analizator

Każdy z nich zostanie dokładnie opisany w kolejnych podrozdziałach.

3.1. Frontend

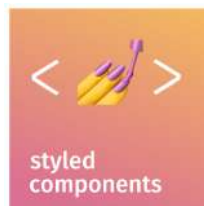
3.1.1. Stos technologiczny

Moduł oparto na frameworku React i języku TypeScript. Oprócz funkcjonalności zapewnianych przez ww. narzędzia wykorzystano również kilka zewnętrznych bibliotek oraz narzędzi:

- Font Awesome, MaterialUI oraz styled-components (patrz rys. 3.4, 3.2 i 3.3) w celu wzbogacenia strony wizualnej projektu
- Eslint oraz Prettier formatujące kod i pilnujące spójności stylu, oraz zachowywania zasad pisania w TypeScriptie
- Axios (patrz rys. 3.5), by uprościć korzystanie z żądań HTTP. Właśnie za ich pomocą, w oparciu o styl architektoniczny REST moduł komunikuje się z Backendem (częścią serwerową)
- Google Maps API wykorzystywane w autorskim narzędziu DrawingTool do tworzenia modeli skrzyżowań



Rysunek 3.2: Logo Material UI



Rysunek 3.3: Logo styled-components



Rysunek 3.4: Logo Font Awesome

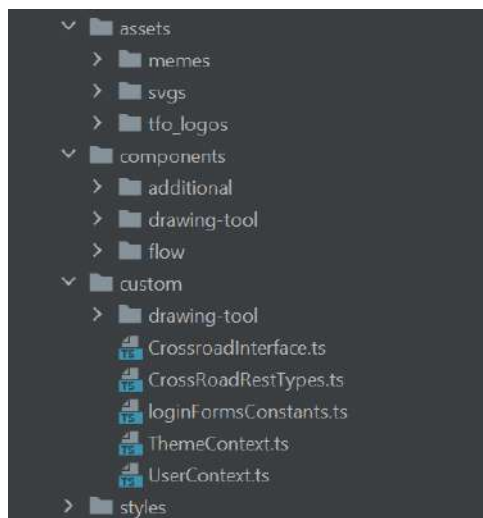


Rysunek 3.5: Logo Axios

Sam React nie wprowadza żadnej konwencji dotyczącej struktury projektu, poza wymaganiem, aby pliki źródłowe znalazły się w katalogu `/src`. Z tego powodu odpowiedzialność za czytelną i efektywną strukturę projektu spada na programistę (patrz rys. 3.6). W projekcie zdecydowano się na podział plików na cztery główne kategorie:

- Assets — grafiki oraz ikony wykorzystywane w module
- Components — pliki zawierające kod JSX, budujące strukturę interfejsu z wykorzystaniem języka HTML, podziałem na tzw. komponenty
- Custom — pliki zawierające stałe, funkcje pomocnicze, spersonalizowane hooki oraz większość niestandardowych typów
- Styles — pliki zawierające klasy styli dla poszczególnych komponentów projektu

Oprócz tego część głównych kategorii posiada podkategorie wydzielone ze względu na przeznaczenie danego komponentu czy jego stylu lub wzajemne podobieństwo tychże komponentów. Dodatkowo najistotniejszy mechanizm modułu, czyli narzędzie `DrawingTool`, posiada własną kategorię we wszystkich głównych kategoriach wyłączając `Assets`.



Rysunek 3.6: Struktura katalogu `/src`

3.1.2. Zaawansowane mechanizmy modułu

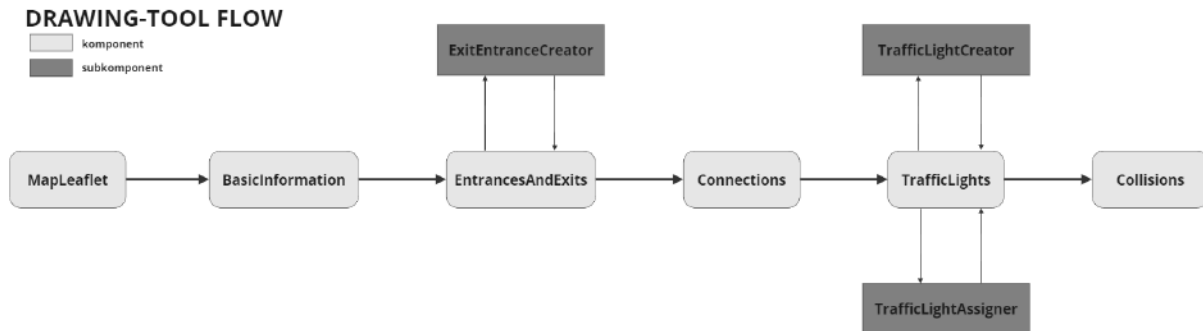
W poprzednich rozdziałach omówiono funkcjonalności zapewniane przez moduł oraz opisano kluczowe z punktu widzenia użytkownika scenariusze korzystania z niego. Komponenty wykorzystywane w tych scenariuszach tworzą również najbardziej zaawansowane mechanizmy tej części projektu i zostaną przybliżone od strony technicznej.

Oba narzędzia wykorzystują żądania HTTP, w celu komunikacji z Backendem, wystawiają kod HTML dla przeglądarki za pomocą rozszerzenia JSX, korzystają z charakterystycznych dla Reacta hooków oraz wykorzystują biblioteki wspierające stylowanie.

DrawingTool

Mechanizm modelowania skrzyżowania jest podzielony na kilka etapów (patrz rys. 3.7), co powoduje istnienie kilku komponentów składających się na cały proces. Wszystkie z nich funkcjonują na tej samej zasadzie i korzystają z tych samych rozwiązań z trzema wyróżnikami, które stosuje tylko część z nich.

1. **Modale** — subkomponenty, które usprawniają proces tworzenia modelu, ułatwiając wprowadzanie informacji o poszczególnych składnikach skrzyżowania. Ich struktura i stosowane tam rozwiązania pokrywają się z tymi wykorzystywanymi we właściwych komponentach. Użytkownik widzi je w formie tzw. modali.
2. **Żądania HTTP** — wykorzystywane tylko przez ostatni komponent, do przesłania modelu skrzyżowania do Backendu, wcześniej powstałe elementy modelu są przekazywane do kolejnych komponentów za pomocą hooka `useNavigate()`.
3. **Google Maps Embed** wykorzystywany w pierwszym komponencie, `MapLeaflet.tsx`, by użytkownik mógł wybrać tło skrzyżowania, które chce zamodelować. Dzięki czemu może je sobie łatwiej wyobrazić i lepiej umiejscowić kolejne jego komponenty.



Rysunek 3.7: Kolejność wykorzystywania kolejnych komponentów i subkomponentów w narzędziu DrawingTool

Komponentem wykorzystującym pierwszy z wyróżników jest *TrafficLights.tsx*. Poniżej znajduje się jego fragment.

```

1  export function TrafficLights() {
2    const navigate = useNavigate();
3    const location = useLocation();
4    const [lightsReady, setLightsReady] = useState(false);
5    const [showLightCreator, setShowLightCreator] = useState(false);
6    const [templateLight, setTemplateLight] = useState(
7      FIRST_STAGE_TRAFFIC_LIGHT_TEMPLATE);
8    const crossroad: Crossroad = location.state.crossroad;
9    useEffect(() => {
10      setCrossroadImage(localStorage.getItem("crossroadMap")!);
11    }, []);
12
13    const onAbort = () => {
14      navigate("../..../crossroad-list");
15      localStorage.removeItem("crossroadMap");
16    };
17    const createTrafficLight = (eeiPointIndex: number) => {
18      setShowLightCreator(true);
19      setTemplateLight({
20        light: {
21          id: "",
22          index: getNewId(firstFreeId, setFirstFreeId),
23          direction:
24            FIRST_STAGE_TRAFFIC_LIGHT_TEMPLATE.light.direction,
25        },
26        eeiPointIndex: eeiPointIndex,
27      });
28    };
  
```

Przykładowe reactowe hooki pozwalające kontrolować stan komponentu oraz funkcje, za pomocą których przetwarzane są akcje wykonywane przez użytkownika.

```

1 <ContainerDiv>
2   {showLightCreator && (
3     <>
4       <TrafficLightCreator
5         closeFunction={() => {
6           onCloseCreator(true);
7         }}
8         handleOnSave={saveFirstStageTrafficLight}
9         trafficLight={templateLight}
10      ></TrafficLightCreator>
11      <Backdrop />
12    </>
13  )}
14  //...
15  <InstructionP>
16    <strong>Traffic Lights</strong>
17    <br />
18    Please follow these steps:
19    <br />
20    //...
21  </InstructionP>
22  <BorderedWorkaroundDiv>
23    {connections.length > 0 &&
24      connections.map((con) => {
25        //...
26        return (
27          <ConnectionMarker
28            key={con.index}
29            thickness={3}
30            entranceX={entrancePoint.xCord}
31            //...
32            color={Colors.BRIGHT_RED}
33            withLightIds={lightsReady}
34            withTooltip={lightsReady}
35            buttonSettings={
36              lightsReady ? buttonSettings : undefined
37            }
38          />
39        );
40      })}
41    {exitEntrancePoints.length > 0 && (
42      <ThemeProvider theme={tooltipTheme}>
43        {exitEntrancePoints.map((point, idx) => {
44          //...
45          const eeiPoint = (
46            <EEIPointMarker
47              key={idx}

```

```

48         color=
49         {matchEEIPointTypeWithColor(point.type)}
50         yCord={point.yCord}
51         xCord={point.xCord}
52     ></EEIPointMarker>
53 );
54
55     return lightsReady ? (
56         eeiPoint
57     ) : (
58         <Tooltip
59             key={idx}
60             title={
61                 <React.Fragment>
62                     <BaseUl>
63                         <BaseLi>
64                             id: {point.index}
65                         </BaseLi>
66                         //...
67                     </BaseUl>
68                     //...
69                 </React.Fragment>
70             }
71             TransitionComponent={Zoom}
72             enterDelay={TOOLTIP_ENTRANCE_DELAY}
73             leaveDelay={TOOLTIP_ENTRANCE_DELAY}
74             arrow
75         >
76             {eeiPoint}
77         </Tooltip>
78     );
79     }}}
80 </ThemeProvider>
81 }}
82 <CrossroadScreenshot
83     src={
84         crossroadImage === undefined
85         ? localStorage.getItem("crossroadMap")!
86         : crossroadImage
87     }
88     alt="Map screenshot"
89 ></CrossroadScreenshot>

```

Część kodu JSX w tym wykorzystanie subkomponentów, stałych z katalogu Custom oraz biblioteki MaterialUI.

Dodawanie video i wybór obszarów

Na poziomie technicznym działanie narzędzia jest zbliżone do DrawingToola, również korzysta z modali, żądań HTTP i bibliotek pomocniczych. Jego wyróżnikiem jest asynchroniczna natura żądania analizy wideo po wyznaczeniu obszarów przejazdów. Dzięki temu analiza odbywa się w tle i użytkownik może w tym czasie wykonywać inne akcje.

```

1  const onFinish = async () => {
2    try {
3      const response = await axios.post<Detection[]>(
4        `/videos/${props.videoId}/analysis`,
5        createdDetectionRectangles,
6        {
7          params: {
8            skipFrames: 10,
9          },
10         headers: {
11           Authorization: `Bearer ${
12             loggedUser !== null
13               ? loggedUser.jwtToken
14               : getUserJWTToken()
15           }`,
16         },
17       },
18     );
19     alert(`Video ${props.videoId} was successfully analyzed!`);
20     setShowSuccessAlert(true);
21     setTimeout(() => {
22       props.onClose();
23     }, 1000);
24   } catch (error) {
25     alert(`Analysis of video ${props.videoId} failed with error
26       ↪ ${error}!`);
27   }

```

Realizacja żądania HTTP z pomocą biblioteki Axios.

```
1 export const AddVideosDiv = styled.div`  
2 width: 100%;  
3 height: 70%;  
4 padding: 0px 30px;  
5 display: flex;  
6 flex-direction: column;  
7 justify-content: space-between;  
8 align-items: center;  
9 `;
```

Przykładowe niestandardowe style stworzone z użyciem styled-components.

Ze względu na obszerność obydwu rozwiązań zaprezentowano jedynie pojedyncze przykłady omawianych rozwiązań użytych w komponentach je tworzących.

3.1.3. Problemy techniczne

Podczas implementacji modułu napotkano niewiele problemów technicznych, jednak na kilka z nich należy zwrócić szczególną uwagę:

- Niezdolność Axiosa do odbierania odpowiedzi na żądanie o typie *image/jpeg*. Biblioteka automatycznie parsuje przesyłane i otrzymywane ciało zapytania oraz odpowiedzi do formatu JSON, co powoduje błędy w obsłudze formatu *.jpg* i *.jpeg*. Z tego powodu, zapytanie, które w odpowiedzi miało przesyłać pierwszą klatkę uprzednio dostarczonego nagrania jest jedynym zrealizowanym w natywnym TypeScript'cie, bez wykorzystania biblioteki.
- Niepoprawne parsowanie list argumentów jako RequestParameters przez Axios — ten sposób przekazywania danych nie jest w stylu REST ściśle ustandaryzowany. Biblioteka przyjęła swój sposób przekazywania tego rodzaju danych, który nie jest akceptowany przez framework Spring po stronie Backendu. Spowodowało to konieczność odpowiedniego przystosowania niektórych endpointów w ww. module.
- Słaba współpraca Eslinta z prettierem oraz styled-components — pierwsze narzędzie, chcąc formatować długość linii kodu przenosiło część opisu stylu do następnej. To z kolei uniemożliwiało drugiej bibliotece poprawne skompilowanie i zastosowanie tegoż stylu. Problem udało się rozwiązać przenosząc kod wykorzystujący styled-components poza pliki *.tsx* i wyłączając Eslinta w tak powstałych plikach, zawierających wyłącznie opisy stylów.
- Skomplikowane edytowanie domyślnych stylów elementów z MaterialUI. Podczas gdy samo zastosowanie komponentów z biblioteki było bardzo proste i znacząco przyspieszyło implementację, konieczność dopasowania ich stylów w zależności od motywu strony oraz wprowadzenie w nich innych drobnych zmian było bardzo uciążliwe i nie zostało dobrze opisane w dokumentacji biblioteki.

3.1.4. Aspekt graficzny

Grafiki loga projektu (patrz rys. 3.8 i rys. 3.9), wykorzystywane w module są w pełni autorskie i wykonane przez autorów projektu. Całe UI oraz UX strony również zostało zaprojektowane w pełni samodzielnie przez członków zespołu, wykorzystując wiedzę nabytą podczas studiów, w szczególności na przedmiocie UX Aplikacji internetowych.



Rysunek 3.8: Logo TFO dla ciemnego trybu



Rysunek 3.9: Logo TFO dla jasnego trybu

3.2. Backend

3.2.1. Stos technologiczny

Serwer aplikacji zaimplementowany jest w Javie przy zastosowaniu Springa. Framework upraszcza tworzenie aplikacji webowych, dzięki umożliwieniu deklaratywnego konfigurowania serwera i jego zależności. Pozwala skupić się na implementacji logiki biznesowej, a jego modularność i czytelność ułatwia współpracę przy tworzeniu kodu.

Za budowanie projektu odpowiada Maven. Dodawanie narzędzi poprzez skopiowanie kilku linijek kodu z Maven Repository [30] pozwala rozbudowywać aplikację wygodnie, dodając funkcjonalności i unikając problemów z konfiguracją oraz bez potrzeby implementacji ich samemu. Przykładowo dodanie zależności OpenCV wygląda następująco:

```

1 <dependency>
2   <groupId>org.openpnp</groupId>
3   <artifactId>opencv</artifactId>
4   <version>4.7.0-0</version>
5 </dependency>
```

3.2.2. Architektura

Komponent Backend łączy wszystkie pozostałe moduły pośrednicząc w komunikacji. W szczególności:

- wystawia endpointy dla Frontendu umożliwiające użytkownikowi sterowanie aplikacją za pomocą intuicyjnego interfejsu graficznego
- tworzy, wysyła i odbiera żądania oraz odpowiedzi od Analizatora i Optymalizatora, po otrzymaniu odpowiednich zleceń
- zapewnia wyłączny dostęp do Bazy Danych
- pośredniczy w komunikacji pomiędzy modułami (żadne dwa inne komponenty nie wchodzi w interakcję bezpośrednią)
- autentyzuje oraz autoryzuje użytkowników

3.2.3. Komponenty

controller

Kontrolery są komponentem frameworka Spring, odpowiadającym za przetwarzanie przychodzących RESTowych żądań. Za pomocą adnotacji można mapować dane adresy URL do odpowiednich klas i funkcji, które obsługują żądanie i zwrócić odpowiedź. Kontrolery mogą także przeprowadzać walidację otrzymanych danych.

```
1 @RestController
2 @CrossOrigin("*")
3 @RequestMapping("/videos")
4 public class VideoController {
5     // ...
6     @GetMapping(value =("/{id}")
7     public ResponseEntity<byte[]> get(@PathVariable String id) {
8         Video video = videoService.getVideo(id);
9
10        if (video != null) {
11            return ResponseEntity
12                .ok()
13                .header(
14                    HttpHeaders.CONTENT_DISPOSITION,
15                    "attachment; filename=\"" + video.getName() + "\""
16                )
17                .body(video.getData());
18        } else {
19            return ResponseEntity
20                .status(NOT_FOUND)
21                .build();
22        }
23    }
24    // ...
25 }
```

Powyższy fragment klasy oznaczony jest adnotacją `@RestController`, więc definiuje kontroler. Używa mechanizmu Cross-Origin Resource Sharing oraz mapuje ścieżki o początku `/videos`. Wybrana funkcja wewnątrz klasy obsługuje żądania HTTP GET o adresie `/videos/id`, gdzie `id` to ID nagrania, którego żądanie ma dotyczyć. W przypadku znalezienia wideo o podanym identyfikatorze funkcja zwróci odpowiedź o kodzie 200 wraz z nagraniem.

service

Serwisy są elementem Springa odpowiedzialnym za implementację logiki biznesowej. W aplikacji dzięki wchodzeniu w bezpośrednią interakcję z repozytoriami stanowią warstwę rozdzielającą je od kontrolerów.

```
1  @Service
2  public class VideoService {
3      // ...
4      public String store(
5          MultipartFile file,
6          String crossroadId,
7          String startTimeId,
8          Integer duration
9      ) {
10         DBObject metadata = new BasicDBObject();
11         metadata.put("crossroadId", crossroadId);
12         metadata.put("startTimeId", startTimeId);
13         metadata.put("type", file.getContentType());
14         metadata.put("duration", duration);
15
16         ObjectId objectId;
17         try {
18             objectId = gridFsTemplate.store(
19                 file.getInputStream(),
20                 file.getOriginalFilename(),
21                 metadata
22             );
23         } catch (IOException e) {
24             return null;
25         }
26
27         return objectId.toString();
28     }
29     // ...
30 }
```

Powyższy kod przedstawia część serwisu służącą do zapisywania nagrania w bazie danych. Klasa oznaczona jest adnotacją *@Service*. Do pliku przypisywane są metadane, a później następuje zapis do bazy. Zwracany jest identyfikator nowododanego obiektu.

document

Bazodanowe dokumenty reprezentowane są przez klasy podobne do poniższej:

```

1  @Document(collection = "users")
2  public class User {
3      @Id
4      private String id;
5
6      @Indexed(unique = true)
7      @NotBlank
8      @Size(min = 4, message = "{validation.name.size.too_short}")
9      @Size(max = 32, message = "{validation.name.size.too_long}")
10     private String username;
11
12     @Email
13     @Indexed(unique = true)
14     @NotBlank
15     private String email;
16
17     @NotBlank
18     private String password;
19
20     private Role role;
21
22     public User(String username, String email, String password) {
23         this.username = username;
24         this.email = email;
25         this.password = password;
26         this.role = role;
27     }
28
29     public String getId() {
30         return id;
31     }
32
33     public void setId(String id) {
34         this.id = id;
35     }
36     // ...
37     // tutaj w kodzie pozostałe gettery i settery
38     // ...
39 }

```

Należy zwrócić uwagę na adnotacje. `@Document(collection = "users")` informuje, że powyższa klasa opisuje dokument należący do kolekcji `users`. `@Id` wiąże atrybut z bazodanowym indeksem, a `"@Indexed(unique = true)"` oznacza, że dana wartość (tutaj nazwa użytkownika oraz email) ma być unikatowa dla kolekcji. Pozostałe adnotacje służą do walidacji dodawanych wartości.

security

Aplikacja wykorzystuje Spring Security. Poniżej przedstawiona jest konfiguracja.

```
1 @Bean
2 public BCryptPasswordEncoder passwordEncoder() {
3     return new BCryptPasswordEncoder();
4 }
```

Używany jest *BCryptPasswordEncoder* [31]. Wybrany został jako kompromis pomiędzy bezpieczeństwem a wykorzystaniem zasobów obliczeniowych [32].

```
1 @Bean
2 public SecurityFilterChain securityFilterChain(
3     HttpSecurity http
4 ) throws Exception {
5     http.cors().and().csrf().disable()
6         .authorizeHttpRequests()
7         .requestMatchers("/auth/**").permitAll()
8         .anyRequest().authenticated()
9         .and()
10        .sessionManagement()
11        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
12
13    http.addFilterBefore(
14        jwtAuthorizationFilter,
15        UsernamePasswordAuthenticationFilter.class
16    );
17
18    return http.build();
19 }
```

Powyższa konfiguracja filtruje przychodzące żądania. Najpierw przetwarzany jest Cross-Origin Resource Sharing (*cors*). Wyłączony jest Cross-site Request Forgery (*csrf*), gdyż używany jest JWT, który stanowi lepszą alternatywę [33]. Następnie w filtrze każdy jest autoryzowany do endpointów o ścieżce *"/auth/**"*, czyli *"/auth/login"* oraz *"/auth/register"*. Żądania do pozostałych zasobów są autentykowane za pomocą JWT.

JWT

Do każdego żądania do Backendu dodawany jest token JWT. Zapisane jest w nim ID użytkownika, co pozwala na zidentyfikowanie wysyłającego żądanie. Token jest aktualny przez ograniczony czas, co zapewnia dodatkowe bezpieczeństwo.

```

1 public String createToken(User user) {
2     Claims claims = Jwts.claims().setSubject(user.getId());
3     Date tokenCreateTime = new Date();
4     Date tokenValidity = new Date(tokenCreateTime.getTime() +
        ↪ TimeUnit.MINUTES.toMillis(accessTokenValidity));
5
6     return Jwts.builder()
7         .setClaims(claims)
8         .setExpiration(tokenValidity)
9         .signWith(SignatureAlgorithm.HS256, secret_key)
10        .compact();
11 }

```

3.2.4. Testowanie

Część funkcjonalności aplikacji pokryta jest testami. W tym celu wykorzystano TestContainers oraz Mockito.

Pierwszy jest frameworkiem do testów integracyjnych. Z jego pomocą można sprawdzić poprawność działania komponentu aplikacji komunikującego się z innymi systemami. Umożliwia on utworzenie Dockerowych kontenerów na podstawie obrazów tych systemów, aby jak najwierniej odwzorować produkcyjne środowisko.

Drugi jest frameworkiem do przeprowadzania tzw. mock testów. Umożliwia tworzenie sztucznych obiektów przypominających rzeczywiste z tą różnicą, że można kontrolować ich zachowanie, aby przetestować działanie wybranego innego obiektu.

Przykładowy test z TestContainers wygląda następująco:

```

1 @Container
2 private static final MongoDBContainer mongoDBContainer =
3     new MongoDBContainer("mongo:6.0")
4     .withExposedPorts(27017);

```

```

1 @DynamicPropertySource
2 static void mongoDbProperties(DynamicPropertyRegistry registry) {
3     mongoDBContainer.start();
4     registry.add("spring.data.mongodb.uri", () ->
        ↪ mongoDBContainer.getReplicaSetUrl() + "?retryWrites=false");
5 }

```

W powyższy sposób naśladowana jest używana w środowisku produkcyjnym baza danych (w celu wykonania testów integracyjnych). Przekazana jest jej nazwa oraz tag obrazu.

```

1 @AfterEach
2 public void cleanUpEach() {
3     userService.getUserRepository().deleteAll();
4 }

```

Po każdym teście dodane rekordy są kasowane. Umożliwia to wykorzystywanie tego samego kontenera pomiędzy wieloma testami.

Przykładowy test serwisu wygląda wówczas następująco:

```

1 @Test
2 public void addUser_properUser_userAdded() {
3     String username = "JDoe";
4     String email = "j.d@gmail.com";
5     String password = "password@123";
6
7     User user = userService.addUser(username, email, password);
8
9     assertEquals(1, userService.getUserRepository().count());
10    assertEquals(username, user.getUsername());
11    assertEquals(email, user.getEmail());
12    assertEquals(password, user.getPassword());
13 }

```

A test z wykorzystaniem Mockito tak:

```

1 @Test
2 void testGetCrossroadsByCreatorIdOrPublic_forMultipleCrossroads() {
3     Crossroad publicCrossroad = new Crossroad(
4         "Grunwaldzkie", "Cracow", "anotherCreator", CrossroadType.PUBLIC,
5         null, null, null, null, null);
6     Crossroad crossroadWithMatchingCreator = new Crossroad(
7         "Grunwaldzkie", "Cracow", "creator123", CrossroadType.PRIVATE,
8         null, null, null, null, null);
9     Crossroad privateCrossroad = new Crossroad(
10        "Grunwaldzkie", "Cracow", "anotherCreator", CrossroadType.PRIVATE,
11        null, null, null, null, null);
12    when(crossroadRepository.findAll()).thenReturn(Arrays.asList(
13        publicCrossroad,
14        crossroadWithMatchingCreator,
15        privateCrossroad
16    ));
17
18    List<Crossroad> result = crossroadService
19        .getCrossroadsByCreatorIdOrPublic("creator123");
20
21    assertEquals(2, result.size());
22 }

```

3.3. Baza Danych

3.3.1. Stos technologiczny

Jako system zarządzania bazą danych wykorzystywane jest MongoDB (patrz rys. 3.10). Jest on nierelacyjny, a obiekty reprezentowane są przez pogrupowane w kolekcjach dokumenty. Przechowywane są one w formacie BSON („Binary JSON”), który wybrano dla efektywności oraz zapewnienia wsparcia dla dat, oraz danych binarnych brakujących natywnie w JSONie. Zaprojektowany został z myślą o skalowalności oraz dowolności projektowania systemu.

Wiadome było, że wraz z rozwojem projektu schemat bazy i struktura obiektów będą ulegać zmianie. Mongo oferuje wygodne rozwiązanie dzięki dużej swobodzie projektowania struktur. Chcąc zostawić sobie otwartą furtkę do umieszczenia aplikacji w chmurze, efektywność horyzontalnej skalowalności Mongo także miała pozytywny wpływ na wybór tego rozwiązania.

Wykorzystano także GridFS, czyli specyfikację służącą przechowywaniu plików. Robi to przy użyciu dwóch kolekcji: jedna odpowiada za metadane, a druga za podzielone na części (tzw. chunki) pliki. Pozwala to przechowywać oraz pobierać zdjęcia i nagrania w znacznie efektywniejszy sposób niż w systemie plików serwera.

Instancja bazy danych uruchamiana jest jako Dockerowy kontener.

3.3.2. Architektura i implementacja

```

1  mongodb:
2      image: mongo:6.0
3      container_name: mongodb
4      networks:
5          - tfo-net
6      ports:
7          - 27017:27017
8      restart: always
9      volumes:
10         - tfo-db:/mongo
11     environment:
12         MONGO_INITDB_ROOT_USERNAME: mongo
13         MONGO_INITDB_ROOT_PASSWORD: *****

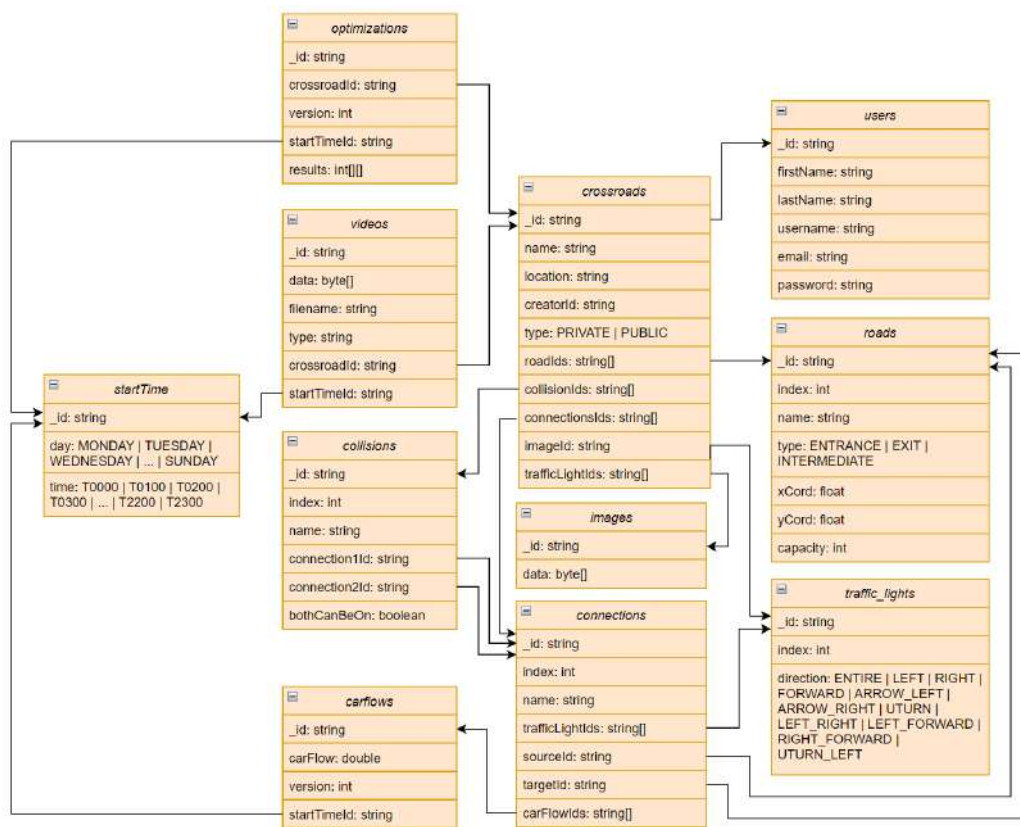
```



Rysunek 3.10: Architektura Bazy Danych

Powyższy kod opisuje konfigurację instancji kontenera Bazy Danych. Zdecydowano się na wybór obrazu *mongo:6.0* [34], gdyż była to sprawdzona wersja, gdy rozpoczynano pracę. Kontener dołączony jest do Dockerowej sieci *tfo-net*, aby otworzyć się na komunikację z Backendem. Port 27017 jest publikowany poza sieć, co umożliwia między innymi podłączenie się do bazy aplikacją MongoDB Compass [35]. Przyłączony jest wolumen pamięci dla persystencji danych w bazie.

3.3.3. Schemat Bazy Danych



Rysunek 3.11: Schemat bazy danych

Dzięki dostarczonym narzędziom na Frontendzie użytkownik nie musi znać tego schematu (patrz rys. 3.11), wszystkie potrzebne dane przekazuje wykorzystując interfejs graficzny. Jest to używana wewnętrznie struktura. Kolekcje nazwane tutaj *images* i *videos* przechowywane są w specyfikacji GridFS, a pole oznaczone w każdej tabeli jako *_id* to identyfikator generowany przez Mongo.

3.4. Optymalizator

3.4.1. Serwer Optymalizatora

Po stronie optymalizatora serwer odbiera zapytania RESTowe wysłane przez Backendowego klienta, na podstawie Pythonowego frameworka FastAPI, zleca optymalizację po czym zwraca jej wyniki.

```

1 app = FastAPI()
2 if __name__ == "__main__":
3     clear()
4     uvicorn.run("Server:app", port=PORT, host="0.0.0.0", reload=True)

```

```

1 @app.post('/optimization')
2 def process_request(request: Request, optimization_request:
  ↳ OptimizationRequestModel):
3     try:
4         basic_optimizer = Optimizer(
5             "../minizinc/models/basic_optimizer_newer.mzn",
6             "../minizinc/models/basic_optimizer_newer_for_comparison.mzn"
7         )
8         optimization_request =
  ↳ OptimizationRequest(optimization_request.optimization_request)
9
10        optimization_request.save_as_dzn(True)
11        optimization_request.save_as_dzn(False)
12
13        data = basic_optimizer.solve(optimization_request, SOLVER)
14        clear(optimization_request.idx)
15    except Exception as error:
16        print(error)
17        return JSONResponse(
18            content={"error_message": "Error occurred during optimization.
  ↳ Possibly invalid data."},
19            status_code=500
20        )
21
22    if data is None:
23        return JSONResponse(
24            content={"error_message": "There are no optimization results
  ↳ for the given time"},
25            status_code=422
26        )
27
28    return JSONResponse(content=data, status_code=200)

```

3.4.2. Optymalizacja

Część zajmująca się dokonywaniem optymalizacji została napisana w języku Minizinc. Kod opisujący jej działanie można podzielić na kilka części, opisanych poniżej.



Ładowanie danych, generowanie struktur i zmiennych

Dane otrzymywane poprzez od modułu Backend po przeparsowaniu są dostarczane do optymalizatora. Na tym etapie również tworzone są potrzebne tablice oraz zmienne pomocnicze

```

1  %%%% Input data
2  int: time_units_in_minute;
3  int: time_unit_count;
4
5  int: light_count;
6  int: road_count;
7  int: connection_count;
8  int: collision_count;
9
10 set of int: Time = 1..time_unit_count;
11 set of int: Timex2 = 1..time_unit_count*2;
12 set of int: Light_ = 0..light_count;
13 set of int: Light = 1..light_count;
14 set of int: Road = 1..road_count;
15 set of int: Connection_ = 0..connection_count;
16 set of int: Connection = 1..connection_count;
17 set of int: Collision = 1..collision_count;
18
19 array[Road] of 0..infinity: road_capacities;
20
21 array[Connection] of int: expected_car_flow;
22 array[Connection] of int: is_connection_from_intermediate;
23 array[Connection, 1..3] of Light_: connection_lights;
24
25 array[Collision] of 0..1: is_collision_important;
26 array[Collision, 1..2] of Connection: collision_connections;
27
28 array[Road, 1..3] of Connection_: road_connections_in;
29 array[Road, 1..3] of Connection_: road_connections_out;
```

Tworzenie funkcji i zmiennych pomocniczych

W celu usprawnienia oraz przyspieszenia solvera zdecydowano się na użycie pewnych zmiennych oraz funkcji pomocniczych, wykorzystywanych później przy wyliczaniu funkcji celu. Użyto również predykatów, w celu szybszego uzyskania rezultatu.

```

1  %%%% Variables
2
3  % whether the light is on
4  array[Light, Time] of var 0..1: is_light_on;
5
6  % whether the connection is passable
7  array[Connection, Time] of var 0..1: is_connection_on;
8  constraint forall(T in Time, C in Connection)(
9      sum(L in [connection_lights[C, L] | L in 1..3 where
10         ↪ connection_lights[C, L]>0])(is_light_on[L, T])<2
11  );
12  constraint forall(T in Time, C in Connection) (is_connection_on[C, T] = (
13      sum(L in [connection_lights[C, L] | L in 1..3 where
14         ↪ connection_lights[C, L]>0])(is_light_on[L, T])
15  ));
16
17  % how much connection is open for cars (what are the chances it will be
18  ↪ chosen)
19
20  array[Connection, Time] of var 0..4: how_much_connection_on;
21  constraint forall(T in Time, C in Connection where is_connection_on[C,
22  ↪ T]=1)(how_much_connection_on[C, T] =
23      4 - sum(L in [connection_lights[C, L] | L in 1..3]) (if L>0 then
24         ↪ is_light_on[L, T] else 0 endif)
25  );
26
27  constraint forall(T in Time, C in Connection where is_connection_on[C,
28  ↪ T]=0)(how_much_connection_on[C, T] = 0);
29
30  % what is the summarized passable connection time
31  array[Connection] of var 0..time_unit_count*4: connection_on;
32  constraint forall(C in Connection)(connection_on[C] = sum(T in
33  ↪ Time)(how_much_connection_on[C, T]));
34
35  % make sure there is only one light_on sequence per light
36  array[Light, Time] of var -1..1: lights_change;
37  constraint forall(L in Light, T in 1..time_unit_count-1)(lights_change[L,
38  ↪ T] = is_light_on[L, T]-is_light_on[L, T+1]);
39  constraint forall(L in Light)(lights_change[L, time_unit_count] =
40  ↪ is_light_on[L, time_unit_count]-is_light_on[L, 1]);
41  constraint forall(L in Light)(count(T in Time)(lights_change[L, T]!=0)<=2);
42  % ...

```

Ograniczenia konieczne do zaspokojenia

Aby solver był w stanie zwrócić rezultat, należy zadeklarować ograniczenia, które konieczne muszą być zaspokojone. Najważniejszym spośród nich, jest brak kolizyjności niektórych spośród świateł. Wykorzystano również mechanizm łamania symetrii w celu uzyskania szybszego rezultatu.

```

1 %%%% Constraints
2
3 % make sure there are no colissions
4 constraint forall(C in Collision, T in Time)
5     (is_connection_on[collision_connections[C, 1],
6       → T]+is_connection_on[collision_connections[C, 2], T]<2);
7
8 % breaking symmetry
9 constraint is_light_on[1, 1] = 1;
10 constraint is_light_on[1, time_unit_count] = 0;
```

Wyliczenie funkcji celu

Aby solver był w stanie zwrócić wynik, konieczne jest również zadeklarowanie funkcji celu, którą próbuje maksymalizować lub minimalizować. Optymalne byłoby wykorzystanie wariacji, ponieważ celem jest uzyskanie jak najbardziej zbliżonych do siebie wartości znormalizowanych przepływów. Ponieważ jednak wykorzystany solver wymusza ograniczenie, aby wszystkie operacje były liniowe względem zmiennych, rozważano skorzystanie ze średniego odchylenia bezwzględnego jako minimalizowanej funkcji celu:

$$\text{średnie odchylenie bezwzględne} = \sum_{c=1}^n \frac{|\text{znormalizowany przepływ}_c - \text{uśredniony przepływ}|}{n}$$

$$\text{uśredniony przepływ} = \frac{\sum_{c=1}^n \text{znormalizowany przepływ}_c}{n}$$

$$\text{znormalizowany przepływ}_c = \frac{\text{liczba pojazdów zdolnych do pokonania danego połączenia}}{\text{liczba pojazdów chętnych na pokonanie danego połączenia}}$$

gdzie n oznacza liczbę połączeń, interpretując połączenie jako przejazd przez skrzyżowanie z konkretnego pasa wjazdowego lub pośredniego na inny, konkretny pas pośredni lub wyjazdowy. *Liczba pojazdów chętnych na pokonanie danego połączenia* zostaje pozyskiwana z danych otrzymywanych z nagrań. *Liczba pojazdów zdolnych do pokonania danego połączenia* jest natomiast uzależniona od cykli świateł.

Jednak po obserwacji rezultatów na etapie testów zauważono, że bardziej efektywnym podejściem okazało się maksymalizowanie minimalnej proporcji. Zdecydowano się więc na maksymalizację poniższej funkcji celu:

$$\min_{\{1 \leq c \leq n\}} (\text{znormalizowany przepływ}_c)$$

```

1  %%%% Objective function
2
3  % counters of connections flows
4  set of float: Flow = 0.0..time_unit_count*4;
5  array[Connection] of var Flow: flow_normalized_no_intermediate;
6  constraint forall(C in Connection where
    ↪ is_connection_from_intermediate[C]=0)
    ↪ (flow_normalized_no_intermediate[C] =
    ↪ connection_on[C]/expected_car_flow[C]);
7  constraint forall(C in Connection where
    ↪ is_connection_from_intermediate[C]=1)
    ↪ (flow_normalized_no_intermediate[C] = 0);
8
9  array[Connection] of var Flow: flow_normalized_intermediate;
10 constraint forall(C in Connection where
    ↪ is_connection_from_intermediate[C]=0) (flow_normalized_intermediate[C]
    ↪ = 0);
11 constraint forall(C in Connection where
    ↪ is_connection_from_intermediate[C]=1) (flow_normalized_intermediate[C]
    ↪ = connection_on[C]/expected_car_flow[C]);
12
13 var Flow: minimum_flow_no_intermediate =
    ↪ min(flow_normalized_no_intermediate);
14 var Flow: minimum_flow_intermediate = min(flow_normalized_intermediate);
15
16 var float: score = minimum_flow_no_intermediate+minimum_flow_intermediate;
17
18 solve maximize score;

```

3.5. Analizator

3.5.1. Serwer Analizatora

Podstawą analizatora nagrań jest serwer w architekturze REST, do którego stworzenia użyty został framework języka Python FastAPI.

```

1 app = FastAPI()
2
3 if __name__ == "__main__":
4     uvicorn.run("Server:app", port=PORT, host="0.0.0.0", reload=True)

```

Serwer wystawia jeden endpoint typu POST o nazwie *analysis_request*. Przyjmuje on obiekt klasy *AnalysisRequest*, który zawiera wideo przekazywane jako string w kodowaniu Base64, parametry analizy obrazu: *skip_frames*, oznaczający, co którą klatkę sieć neuronowa powinna analizować, oraz *detection_rectangles*, definiujące pola zliczające przejeżdżające pojazdy. Poza tym przyjmuje ID przekazywanego nagrania oraz rozszerzenie, z jakim powinno ono być zapisane.

```

1 @app.post("/analysis")
2 def analysis_request(
3     analysis_request: AnalysisRequest) -> str:
4
5     file_name = VIDEOS + analysis_request.id + "." +
6         ↪ analysis_request.extension
7
8     car_counter = CarCounter("yolo", VIDEOS + analysis_request.id + '.' +
9         ↪ analysis_request.extension,
10         ↪ "output", int(analysis_request.skip_frames),
11         ↪ analysis_request.detection_rectangles,
12         ↪ analysis_request.video)
13
14     count_cars = car_counter.run()
15
16     return count_cars

```

```

1 @app.post("/analysis")
2 class AnalysisRequest(BaseModel):
3     id: str
4     extension: str
5     skip_frames: str
6     detection_rectangles: List[DetectionRectangle]
7     video: str

```



```
1 from pydantic import BaseModel
2 from Pair import Pair
3
4
5 class DetectionRectangle(BaseModel):
6     connectionId: str
7     lowerLeft: Pair
8     upperRight: Pair
9     detected_car_ids = set()
10    detected_bus_ids = set()
```

3.5.2. Wykrywanie pojazdów

Wykrywanie pojazdów odbywa się z wykorzystaniem wytrenowanego wcześniej modelu. Wagi oraz kod użytego przez nas modelu został pobrany z publicznego [repozytorium na Githubie](#).

Częstotliwość wykrywania pojazdów przez model zdefiniowana jest przez wymieniony wyżej parametr *skip_frames*. Sieć używana jest tylko raz na *skip_frames* klatek. W pozostałych pojazdy śledzone są przez *correlation_tracker* z biblioteki *dlib*. Śledzenie wykrytych obiektów zamiast wykrywania ich w każdej klatce pozwala na znaczne przyspieszenie analizy obrazu oraz zminimalizowanie ryzyka kilkukrotnego wykrycia jednego pojazdu w obrębie tego samego *detection_rectangle*.

Organizacja pracy

4.1. Charakterystyka projektu

Legenda

- akcje
- planowanie
- notatki
- rozpoczęto tworzenie nowego scenariusza
- jaka kolejność
- jaka metyka

Frontend: React

- rozpoczęto tworzenie nowego scenariusza
- jaka kolejność
- jaka metyka

Backend: Java+Spring

- rozpoczęto tworzenie nowego scenariusza
- jaka kolejność
- jaka metyka

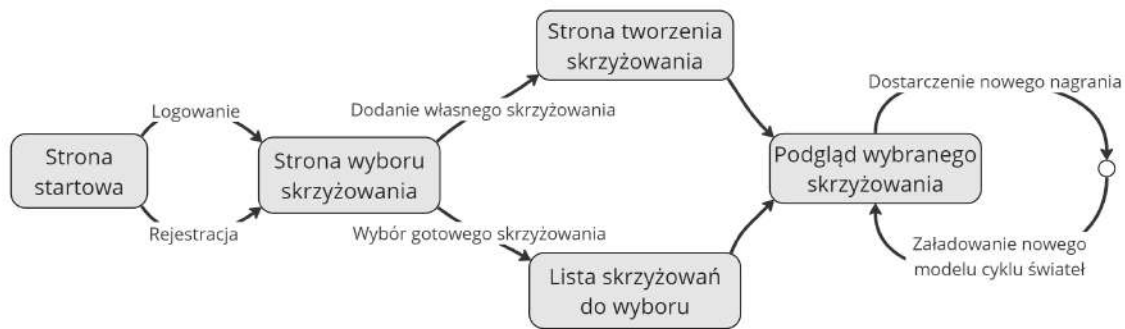
Optymalizator: Python + Minizinc

- rozpoczęto tworzenie nowego scenariusza
- jaka kolejność
- jaka metyka

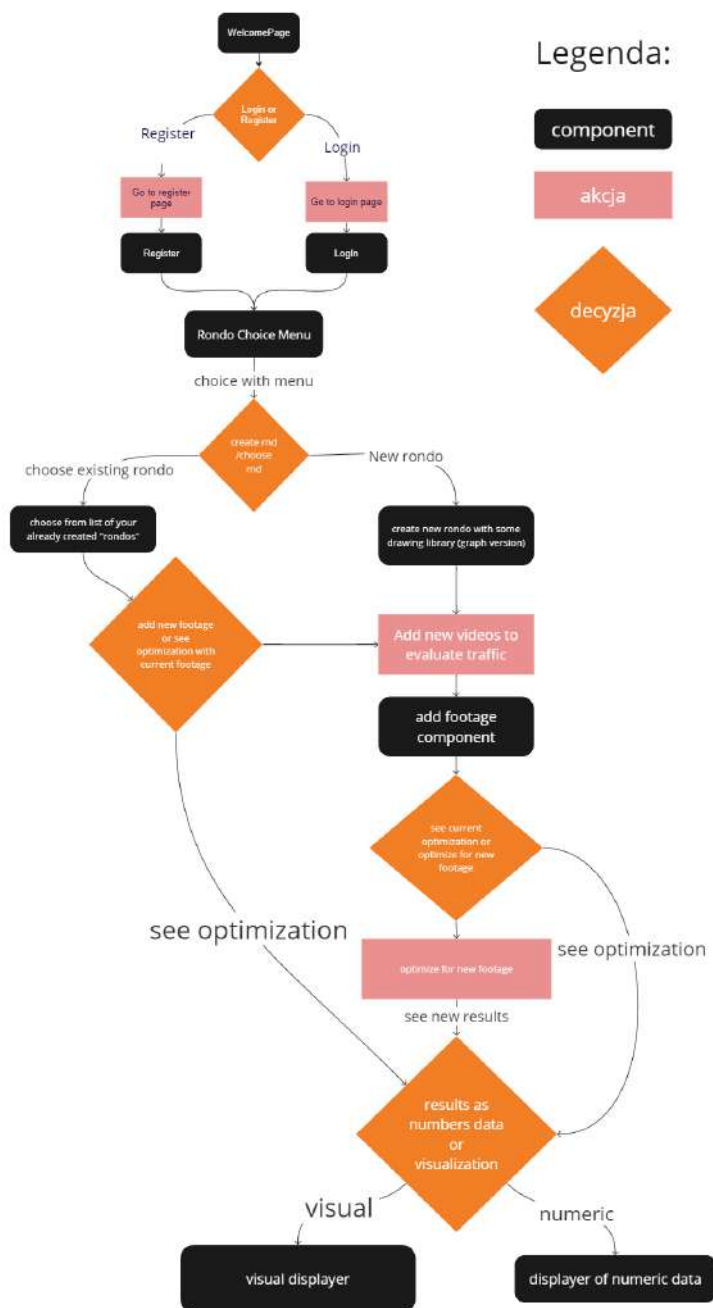
DataBase: Tekstowa

- rozpoczęto tworzenie nowego scenariusza
- jaka kolejność
- jaka metyka

W początkowych tygodniach pracy, podczas jednego ze spotkań określono poprzez rozrysowanie schematów (patrz rys. 4.2 i 4.3), w jaki sposób, w przybliżeniu, powinna wyglądać aplikacja ze strony użytkownika, zarówno ze strony koncepcyjnej jak i implementacyjnej.



Rysunek 4.2: Wstępna koncepcja przepływu Frontendu



Rysunek 4.3: Uszczegółowienie powyższej koncepcji

Rzeczywiście rozwój projektu był procesem iteracyjnym. Przez początkowe miesiące pracy spotkania zespołowe odbywały się co tydzień, w celu nadzorowania postępów pracy, jednak z czasem okazało się, że rzadsze spotkania kontrolne są wystarczające i proces rozwoju aplikacji przyjął formę zbliżoną do metodyki Kanban. Mimo tego utrzymywaliśmy komunikację na zadowalającym poziomie, ponieważ informowaliśmy się wzajemnie o postępach przez komunikatory tekstowe. Było to szczególnie istotne, gdy kilka osób pracowało nad tym samym elementem aplikacji. W takich przypadkach często dochodziło do blokady postępów pracy bądź rozbieżności w wersjach kodu, jednak z pomocą systemu kontroli wersji Git [36] udawało nam się rozwiązywać je bez większych problemów.

4.2. Podział zadań

Nad wieloma komponentami aplikacji członkowie zespołu pracowali razem, więc sposób podziału nie jest oczywisty. Przykładowo wszyscy wspólnie projektowali schemat bazy danych. Można jednak wyróżnić obszary aplikacji, nad którymi dana osoba pracowała więcej niż reszta zespołu.

Nikodem Korohoda

- Komponent optymalizator
- Współpraca nad endpointem do optymalizacji w Backendzie
- Symulator wyniku optymalizacji
- Prezentacja wyników w postaci symulacji
- Koordynacja rozwoju projektu

Mateusz Więcek

- Komponent Analizator
- Współpraca nad endpointem do analizy wideo w Backendzie
- Uwierzytelnienie i autoryzacja użytkowników
- Elementy Backendu

Jędrzej Ziebur

- Komponent Frontend
- Autorskie narzędzie DrawingTool
- Design logo aplikacji
- UI oraz UX strony klienckiej produktu

Szymon Żychowicz

- Wirtualizacja aplikacji — implementacja plików Dockerfile oraz docker-compose.yaml dla modułów aplikacji, oraz zapewnienie komunikacji między nimi
- Komponent Backend i Database
- Pomoc w implementacji części serwerowej w Analizatorze i Optymalizatorze
- Przygotowanie testów do Backendu

4.3. Wykorzystane narzędzia

4.3.1. Komunikacja

Do komunikacji zostały wykorzystane:

- Facebook Messenger [37] — do szybkiego kontaktu i ustalania spraw organizacyjnych
- Discord [38] — do przesyłania fragmentów kodu, przykładowych danych do testów, do spotkań oraz do udostępniania ekranu, gdy trzeba coś zaprezentować grupie

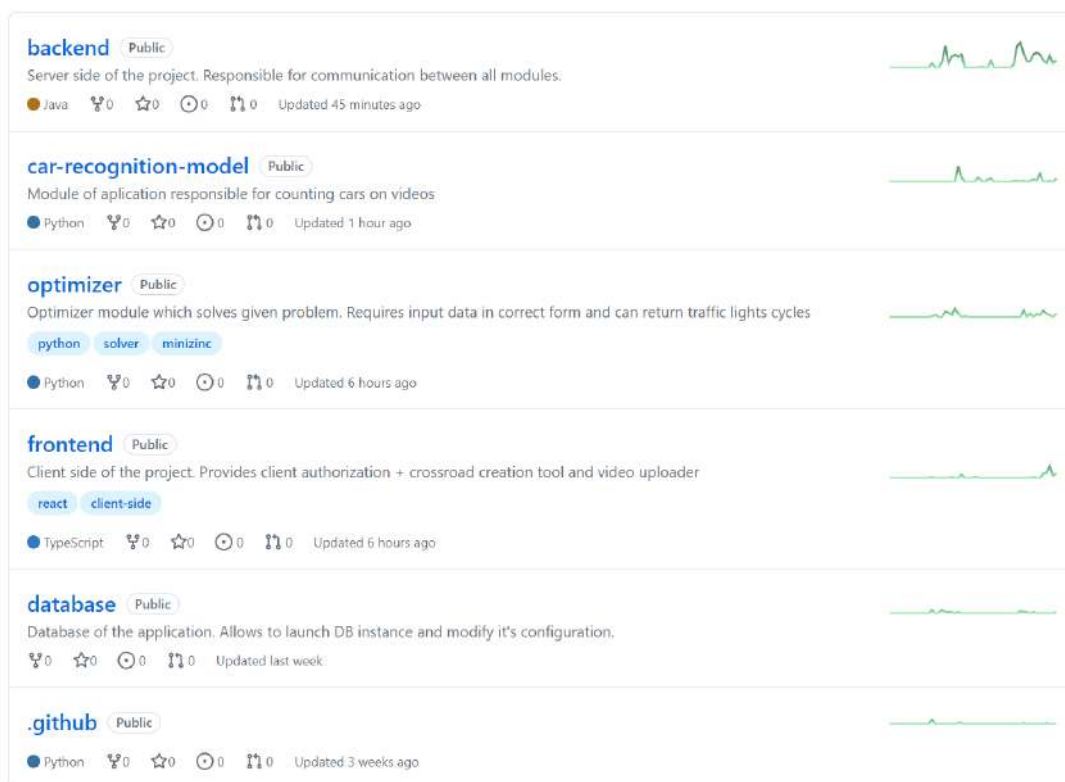


4.3.2. Współpraca

Git jest niepodważalnie podstawowym narzędziem do grupowej pracy nad kodem. Bez systemu kontroli wersji cały czas powstawałyby konflikty w kodzie paraliżujące rozwój projektu, dlatego był to oczywisty wybór.

Repozytorium kodu umieszczono na platformie Github [39]. Utworzona została organizacja, gdyż bardzo szybko okazało się, że monorepo zawierające wszystkie moduły projektu nie jest optymalnym rozwiązaniem. Organizacja składa się z repozytoriów reprezentujących podstawowe moduły aplikacji oraz jednego repozytorium organizacyjnego, zawierającego README projektu oraz skrypt do zarządzania repozytoriami (patrz rys. 4.4). Warto również wspomnieć, że na przestrzeni całego procesu tworzenia aplikacji korzystano z Branchów i funkcji Pull Request oraz członkowie zespołu wystawiali sobie wzajemnie Review tak, aby wspólnie dbać o dobrą jakość tworzonego kodu.





Rysunek 4.4: Organizacja repozytoriów na GitHubie

Przez brak dostępu do Jiry zdecydowano się utworzyć arkusz w Google Sheets pełniący podobne funkcje (patrz rys. 4.5). Tam wpisywano zadania, ich priorytety, osoby odpowiedzialne za nie oraz status ich wykonania.



Rysunek 4.5: Lista zadań w arkuszu Google Sheets

Podczas pracy nad projektem spotkania twarzą w twarz były rzadkością. Do projektowania aplikacji nie było możliwe wykorzystywanie kartki papieru i długopisu, ale dzięki Miro [40] nie było to problemem. Strona ta pozwala wielu osobom w tym samym czasie rysować i pisać na arkuszu przypominającym szkolną tablicę.



4.3.3. Przebieg pracy

Praca nad projektem rozpoczęła się w marcu 2022 roku. Przebiegała ona względnie stałym tempem poza przerwą wakacyjną. W retrospekcji można wyznaczyć trzy etapy prac:

- I etap — od początku pracy do wakacji
- II etap — wakacje
- III etap — koniec wakacji do końca projektu

W pierwszym etapie prac powstawały początkowe, ogólne idee dotyczące zakresu funkcjonalności aplikacji oraz zadań wyodrębnionych komponentów. Praca nad modułami przebiegała względnie niezależnie od siebie. Celem zespołu było przygotowanie narzędzi zdolnych do wykonywania swoich obowiązków na zmockowanych danych.

W drugim etapie powstawał DrawingTool oraz analizator nagrań. DrawingTool to duże, złożone narzędzie, stworzone od podstaw przez zespół, dlatego praca nad nim wymagała dużych nakładów czasu. Analizator jest mniejszym komponentem, więc głównym zadaniem związanym z jego implementacją było zintegrowanie pobranego z GitHuba programu z naszym systemem oraz optymalizacja i przyspieszenie jego działania. Rozwój pozostałych komponentów w tym czasie był znikomy.

W trzecim etapie priorytetem było połączenie przygotowanych modułów. Osobno zapewniały zadowalające działanie, jednak aby aplikacja mogła funkcjonować, musiały zacząć współpracować, stąd nacisk na implementację i zmiany RESTowych endpointów oraz dopracowywanie wysyłania i przetwarzania żądań HTTP w każdym z modułów aplikacji. Proces ich łączenia zmuszał do wprowadzania wielu zmian wynikających z różnic w interfejsach powstałych przez uprzedni niezależny rozwój. Podczas tego etapu dokończano również niektóre funkcjonalności z poszczególnych modułów, w szczególności narzędzie wyboru obszarów przejazdów na nagraniach ruchu oraz optymalizator. Dodatkowo w tym okresie zwiększono nacisk na testowanie i wirtualizację aplikacji. Ulepszono również prezentację wyników za pomocą danych opisowych, oraz stworzono symulację pomagającą w porównaniu rezultatów optymalizacji.

Rozdział 5

Wyniki projektu

Jako wyniki projektu można uznać zarówno aplikację, jak i efekty jej zastosowania, czyli zoptymalizowane cykle świateł. W poniższym rozdziale oba aspekty zostaną kolejno omówione, a także zostaną przedstawione możliwości dalszego rozwoju oraz podsumowanie projektu ze strony autorów.

5.1. Uzyskany produkt

Oprogramowanie

Dostarczona aplikacja zapewnia wszystkie założone funkcjonalności, łącząc w sobie możliwości wszystkich pięciu modułów, napisanych w czterech językach programowania, wykorzystujących wiele bibliotek oraz frameworków React, Spring i CBC Coin.

Dokumentacja

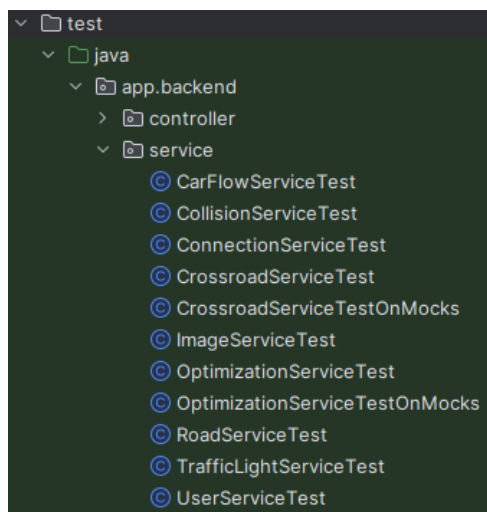
Część webowa aplikacji została zaprojektowana z myślą o jak największej przejrzystości i prostocie użytkowania. Dodatkowo każdy etap procesu modelowania skrzyżowania, dodawania nagrania, zlecenia optymalizacji oraz podglądu wyników opatrzony jest krótką instrukcją postępowania. Rozwiązanie jest typu Open Source, a moduły Backend, Optymalizator oraz Analizator wystawiają swoje API dzięki możliwościom Springa oraz FastAPI.

Wdrożenie

Wszystkie moduły z wyjątkiem Frontendu zostały zwirtualizowane do Dockerowych kontenerów, co umożliwia ich wygodne uruchomienie na własnym sprzęcie. Należy jednak dodać, że w wersji zwirtualizowanej Backend nie zwraca pierwszej klatki przesyłanego nagrania z powodu problemów z biblioteką OpenCV.

Pokrycie testami

Moduł backendowy został pokryty testami jednostkowymi i integracyjnymi (patrz rys. 5.1), testującymi m.in. połączenie z innymi modułami, w szczególności bazą danych. Kontrola jakości modułu frontendowego była przeprowadzana ręcznie w formie testów akceptacyjnych niezależnie przez wszystkich członków zespołu.

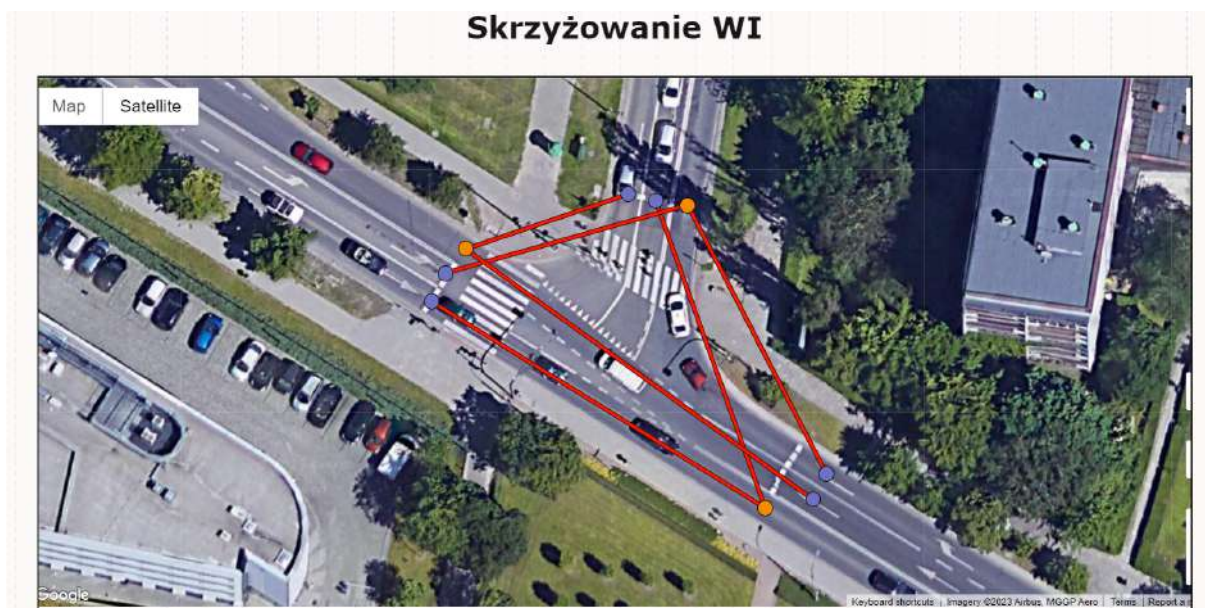


Rysunek 5.1: Zestaw testów modułu Backend

5.2. Efekty działania aplikacji

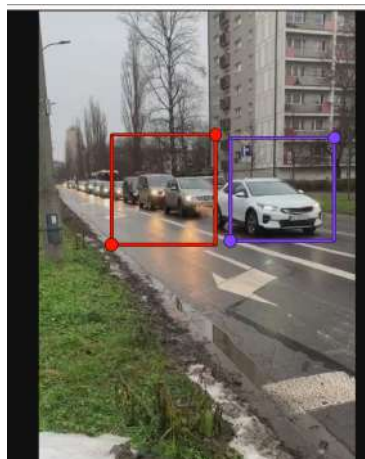
Głównymi celami aplikacji było zapewnienie użytkownikowi wygodnego sposobu na zamodelowanie wybranego skrzyżowania, dostarczeniu nagrań ruchu i optymalizacji cykli świateł za pomocą optymalizatora. Proces ten łączy wszystkie opisane scenariusze użytkownika w jedną spójną sekwencję operacji. Ponieważ wymagania niefunkcjonalne zostały dokładnie opisane w poprzednich rozdziałach, w tej sekcji skupiono się na analizie skuteczności działań optymalizatora, czyli czy przedstawił on cykl świateł faktycznie zwiększający przepływ samochodów na skrzyżowaniu, a jeśli tak to, jakiej poprawy dokonał i ile klient musiał czekać na efekt obliczeń.

Jako modelowy obiekt skrzyżowania przyjęto spotkanie ulic Czarnowiejskiej, Nawojki oraz al. Kijowskiej nieopodal budynku Wydziału Informatyki AGH w Krakowie (patrz rys. 5.2). Do aplikacji wprowadzono następującą reprezentację:

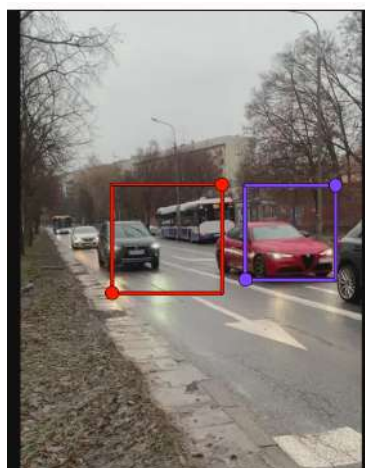


Rysunek 5.2: Model skrzyżowania

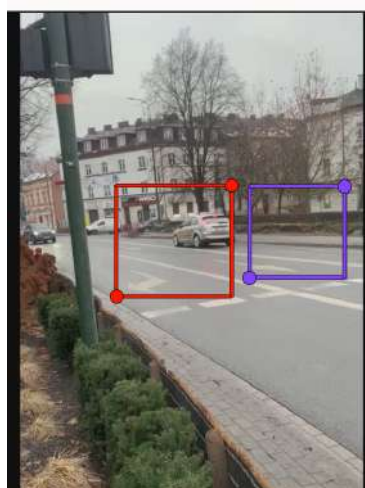
Następnie, około godziny 8:20, dla każdej z wjazdowych ulic wykonano 7 minutowe nagrania, które następnie dostarczono do aplikacji, określając odpowiednio pasy wjazdowe i wyjazdowe (patrz rys. 5.3, 5.4 i 5.5).



Rysunek 5.3: Pierwsza klatka nagrania od strony alei Kijowskiej



Rysunek 5.4: Pierwsza klatka nagrania od strony ulicy Nawojki



Rysunek 5.5: Pierwsza klatka nagrania od strony ulicy Czarnowiejskiej

Po przeanalizowaniu nagrań przez moduł Car Recognition, poprzez aplikację zlecono optymalizację cykli świateł dla dnia i godziny, dla których dostarczono nagranie (patrz rys. 5.6). Na to zadanie nie przydzielano optymalizatorowi ograniczenia czasowego. Wynik został otrzymany po niecałych pięciu sekundach.

Crossroad: Skrzyżowanie WI

Optimization Time:

no time limit

Hour and day to optimize:

08:00

WEDNESDAY

Close Start optimization

Rysunek 5.6: Zlecenie optymalizacji i jego parametry

Efekty działania modułu oceniono poprzez porównanie wartości dwóch metryk, sprzed oraz po optymalizacji:

1. porównania liczby samochodów przejeżdżających przez poszczególne połączenia dla cykli świateł
2. porównania proporcji liczby samochodów względem czasu świecenia się zielonego światła dla cykli świateł

Porównania liczb samochodów

Wykorzystując tę metrykę, porównywane są liczby samochodów, które są w stanie pokonać dane połączenie w ciągu minuty dla cykli świateł sprzed oraz po optymalizacji. W tabeli użyto skrótu „LS”, aby oznaczyć liczbę samochodów (patrz tab. 5.1).

Tabela 5.1: Porównanie liczb samochodów

Nazwa przejazdu	LS pierwotnie przepuszczona	LS przepuszczona po optymalizacji	Zysk w samochodach
Nawojki → Czarnowiejska	7	7	0
Nawojki → Kijowska	5	10	5
Czarnowiejska → Nawojki	11	11	0
Czarnowiejska → Kijowska	1	1	0
Kijowska → Czarnowiejska	5	5	0
Kijowska → Nawojki	1	1	0

Można zauważyć, że w większości przejazdów liczba samochodów nie uległa poprawie. Wynika to z faktu, że dla godziny przeprowadzania testu ruch samochodowy był niewielki. Jednak, pomimo tego, jedno z połączeń i tak zostało usprawnione.

Porównania proporcji

Jako drugą miarę zdecydowano się wykorzystać proporcje liczb samochodów w stosunku do długości świecenia się zielonych świateł (patrz tab. 5.2). Jest to powodowane faktem, że jednym z zadań aplikacji jest dopasowanie świateł do ruchu w taki sposób, aby czas świecenia się zielonego światła był proporcjonalny do natężenia ruchu na poszczególnych pasach.

Wzór na wyliczanie stosunku jest następujący:

$$\frac{\text{Liczba sekund przez jaką połączenie jest przejezdne}}{\text{Średnia liczba nadjeżdżających samochodów w ciągu minuty}}$$

Tabela 5.2: Porównanie stosunków samochodów

Nazwa przejazdu	Stosunek sprzed optymalizacji	Stosunek po optymalizacji
Nawojki → Czarnowiejska	7	4,3
Nawojki → Kijowska	0,5	2,4
Czarnowiejska → Nawojki	4	2,2
Czarnowiejska → Kijowska	52	6
Kijowska → Czarnowiejska	1,6	2,4
Kijowska → Nawojki	54	15

Jak można zauważyć, stosunki znacząco się do siebie zbliżyły. Potwierdza to również porównanie wartości wariancji (patrz tab. 5.3).

Tabela 5.3: Porównanie wariancji stosunków

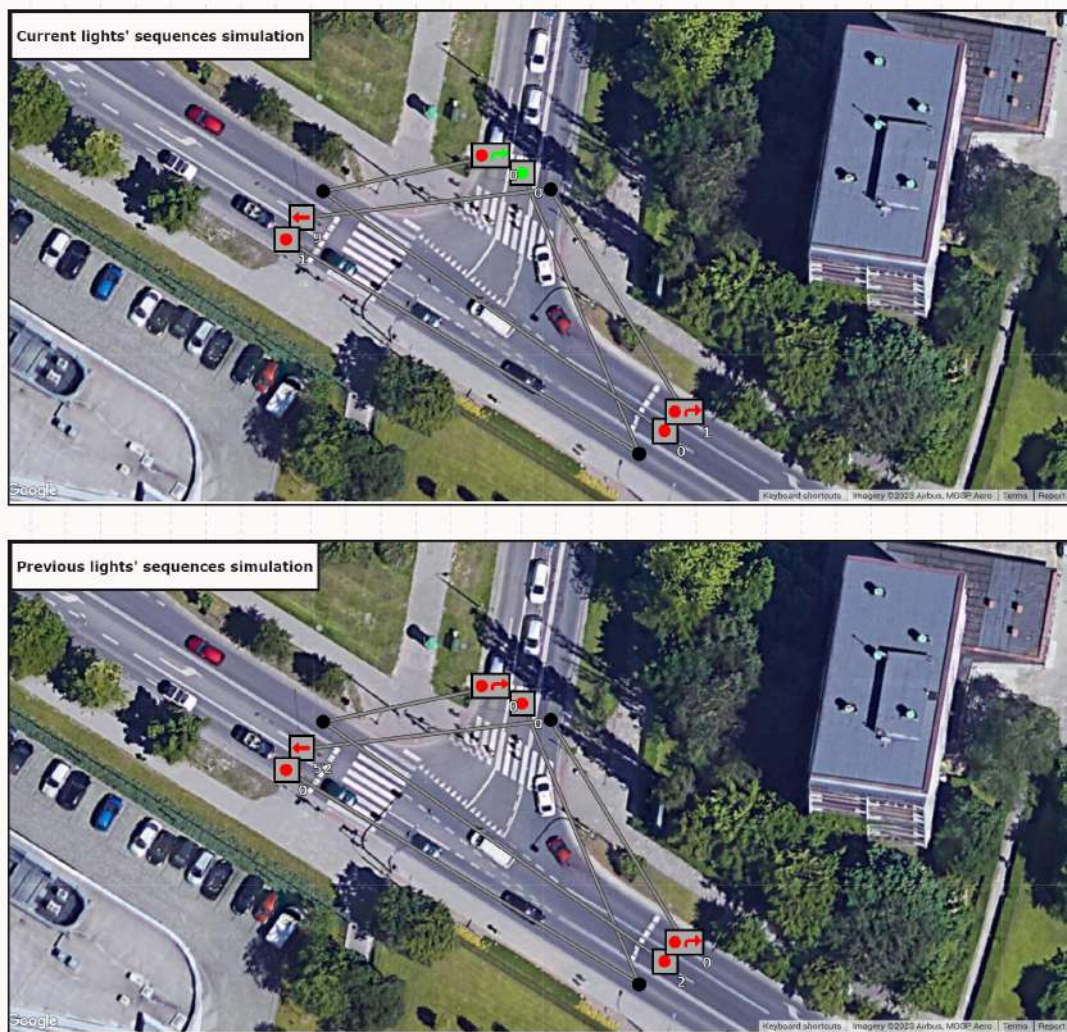
Wariancja stosunków sprzed optymalizacji	Wariancja stosunków po optymalizacji
664,74	24,42

Znaczący spadek wartości wariancji implikuje istotną poprawę wyników. Oznacza to bowiem, że wartości stosunków długości świecenia świateł względem liczby nadjeżdżających samochodów są bardziej do siebie zbliżone.

Porównanie wyników można zobaczyć w postaci danych opisowych (patrz rys. 5.7) lub symulacji (patrz rys. 5.8).



Rysunek 5.7: Fragment wyników w formie opisowej



Rysunek 5.8: Stan symulacji po czasie odpowiadającym 10 minutom

5.3. Ograniczenia rozwiązania

Od początku prac zespół był świadomy, iż przygotowywane rozwiązanie jest bliższe badaniu potencjału produktu, tzw. Proof of Concept (PoC), niż w pełni kompletnemu systemowi, zarówno ze względu na nakłady finansowe, jakie zostały przeznaczone na komercyjne rozwiązania, jak i na zastosowane metodyki. Pomimo niezaprzeczalnej wartości merytorycznej aplikacji, posiada ona niedociągnięcia, które uniemożliwiają w pełni skuteczne stosowanie rozwiązania w każdej możliwej sytuacji i powinny być brane pod uwagę przy potencjalnym dalszym rozwijaniu produktu, a są to:

- Niebranie pod uwagę pasów, na które wstęp mają tylko określone rodzaje pojazdów, w szczególności bus pasów
- Nieuwzględnianie żółtych świateł
- Skalowanie długości cykli świateł do minuty
- Nieuwzględnianie przejazdów dla pociągów, tramwajów tudzież premetra
- Ignorowanie zmian sygnalizacji wymuszonych ruchem pieszych i rowerów

- Ujednolicanie czasu pokonania danego połączenia przez pojazd niezależnie od jego rodzaju i pokonywanego dystansu
- Konieczność dostarczenia bardzo jakościowych i specyficznych pod względem kąta nagrania materiałów wideo
- Niedoskonałość modelowania i prezentacji skrzyżowania w webowej części aplikacji
- Skupienie uwagi rozwiązania na pojedynczym węźle komunikacyjnym, zamiast na całej ich sieci

5.4. Propozycje dalszego rozwoju

Jeśli potencjał produktu oceniono by pozytywnie, pierwszym krokiem dalszego rozwoju powinno być zaadresowanie problemów wymienionych w poprzedniej sekcji, należących do problemów natury funkcjonalnej rozwiązania. Kolejne aspekty, na które należałoby zwrócić uwagę, to:

- Zwiększenie szybkości skutecznej analizy nagrania przez moduł Car Recognition
- Poprawa płynności niektórych operacji webowej części aplikacji, tj. dodawanie nowego światła czy przypisywanie go do przejazdu
- Ulepszenie UX części webowej
- Przeniesienie optymalizatora do frameworka Google OR-Tools
- Reewaluacja schematu bazy danych
- Naprawa pojedynczych błędów i unifikacja osobnych wirtualizacji modułów do pojedynczego Docker Compose

Można je zaklasyfikować do optymalizacji działającego już systemu.

5.5. Podsumowanie

W opinii zespołu projekt można uznać za udany. Produkt ma pewne niedoskonałości, oraz nie realizuje wszystkich założeń, które przyjęto w pierwszych tygodniach pracy, jednakże biorąc pod uwagę jego skalę oraz możliwości czasowe i techniczne efekt końcowy jest zadowalający. Proces tworzenia aplikacji zwiększył świadomość twórców odnośnie znaczenia dobrej organizacji pracy, precyzyjnego specyfikowania zadań koniecznych do implementacji funkcjonalności oraz planowania rozwoju projektu dla osiągnięcia satysfakcjonujących rezultatów. Początkowo przywiązywano dużą uwagę do zachowywania wyżej wymienionych aspektów, jednak z czasem gdy przybywało zadań, utrzymanie wystarczająco wysokiego poziomu koordynacji pracy stało się uciążliwe, jako że poświęcano więcej czasu i uwagi na czystą implementację kosztem analizy koncepcyjnej. Mogło to wynikać z niewystarczającej liczby osób w zespole jak na tak złożony projekt.

Bibliografia

- [1] Eurostat. *Persons in employment by commuting time, educational attainment level and degree of urbanisation*. https://ec.europa.eu/eurostat/databrowser/view/LFSO_19PLWK28/bookmark/table?lang=en&bookmarkId=96ad2e59-267b-404c-97e2-133b7fa05022. Online data code: LFSO_19PLWK28.
- [2] Eurostat. *Employed people by commuting time and country*. <https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20201021-2>. Online data code: LFSO_19PLWK28.
- [3] K. Zhang i S. Batterman. „Air pollution and health risks due to vehicle traffic”. W: *Science of The Total Environment* 450-451 (2013), s. 307–316. DOI: <https://doi.org/10.1016/j.scitotenv.2013.01.074>. URL: <https://www.sciencedirect.com/science/article/pii/S0048969713001290>.
- [4] European Environment Agency. „Managing exposure to noise in Europe”. W: (2017). URL: <https://www.eea.europa.eu/publications/managing-exposure-to-noise-in-europe/noise-in-europe-updated-population-exposure>.
- [5] S. Bharadwaj, S. Ballare, Rohit i M. K. Chandel. „Impact of congestion on greenhouse gas emissions for road transport in Mumbai metropolitan region”. W: *Transportation Research Procedia* 25 (2017). World Conference on Transport Research - WCTR 2016 Shanghai. 10-15 July 2016, s. 3538–3551. DOI: <https://doi.org/10.1016/j.trpro.2017.05.282>. URL: <https://www.sciencedirect.com/science/article/pii/S2352146517305896>.
- [6] *Obwieszczenie ministra infrastruktury oraz ministra spraw wewnętrznych i administracji z dnia 31 października 2019 r. w sprawie ogłoszenia jednolitego tekstu rozporządzenia Ministrów Infrastruktury oraz Spraw Wewnętrznych i Administracji w sprawie znaków i sygnałów drogowych*. URL: <https://isap.sejm.gov.pl/isap.nsf/download.xsp/WDU20190002310/O/D20192310.pdf>.
- [7] *Vehicle Tracking Using OpenCV and VOLOv5 (VTUOV)*. URL: <https://github.com/nathen418/Vehicle-Tracking-Using-OpenCV-and-VOLOv5>.
- [8] *Polska strona firmy PTV Group*. URL: <https://www.ptvgroup.com/pl/rozwiazania/produkty/ptv-balance-ptv-epics/>.
- [9] *Strona firmy PTV Group w języku angielskim*. URL: https://www.myptv.com/en/mobility-software/adaptive-traffic-control-system-ptv-balance-epics?_ga=2.41184674.1676361527.1682419017-799028602.1681578380.
- [10] *Spis produktów firmy Gevas*. URL: <https://crossig.gevas.eu/pl/produkty/>.
- [11] *Strona firmy OneRoad, wykorzystującej PTV Vissim*. URL: <http://www.oneroad.pl/vissim/>.

-
- [12] *Obszarowy system sterowania ruchem i nadawanie priorytetu dla transportu zbiorowego w Krakowie*. URL: <https://edroga.pl/inzynieria-ruchu/obszarowy-system-sterowania-ruchem-i-nadawanie-priorytetu-dla-transportu-zbiorowego-w-krakowie-cz-i-27089069>.
 - [13] *Inteligentne systemy przyspieszają komunikację*. URL: https://www.bip.krakow.pl/?news_id=65131.
 - [14] *Strona firmy Isarsoft*. URL: <https://www.isarsoft.com/solutions/traffic>.
 - [15] *Strona producenta Dockera*. URL: <https://www.docker.com>.
 - [16] *Strona producenta Maven*. URL: <https://maven.apache.org>.
 - [17] *Strona producenta Spring*. URL: <https://spring.io>.
 - [18] *Strona producenta Testcontainers*. URL: <https://testcontainers.com>.
 - [19] *Strona producenta Mockito*. URL: <https://site.mockito.org>.
 - [20] *Strona producenta MongoDB*. URL: www.mongodb.com/.
 - [21] *Strona producenta FastAPI*. URL: <https://fastapi.tiangolo.com>.
 - [22] *Strona producenta OpenCV*. URL: <https://opencv.org>.
 - [23] *Podręcznik użytkownika CBC*. URL: <https://www.coin-or.org/Cbc/>.
 - [24] *Strona fundacji COIN-OR*. URL: <https://www.coin-or.org/>.
 - [25] *Strona producenta Gecode*. URL: <https://www.gecode.org/>.
 - [26] *Dokumentacja Google Maps Embed API*. URL: <https://developers.google.com/maps/documentation/embed/get-started?hl=en>.
 - [27] *Strona producenta Reacta*. URL: <https://react.dev>.
 - [28] *Strona biblioteki styled-components*. URL: <https://styled-components.com/>.
 - [29] *Strona biblioteki Material UI*. URL: <https://mui.com>.
 - [30] *Strona Maven Repository*. URL: <https://mvnrepository.com>.
 - [31] *Dokumentacja enkodera*. URL: <https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoder.html>.
 - [32] *Porównanie enkoderów haseł*. URL: <https://saurabhchaudhary01.medium.com/comparing-password-encoders-in-spring-security-42f88c83cb26>.
 - [33] *Porównanie JWT i CSRF*. URL: <https://www.linkedin.com/pulse/securing-your-api-stack-benefits-replacing-csrf-tokens-mcgowan/>.
 - [34] *Strona MongoDB na Docker Hub*. URL: https://hub.docker.com/_/mongo.
 - [35] *Strona producenta MongoDB Compass*. URL: <https://www.mongodb.com/products/tools/compass>.
 - [36] *Git*. URL: <https://git-scm.com/>.
 - [37] *Messenger*. URL: <https://www.messenger.com/>.
 - [38] *Discord*. URL: <https://discord.com/>.
 - [39] *GitHub TrafficFlowOptimizer*. URL: <https://github.com/TrafficFlowOptimizer>.
 - [40] *Miro*. URL: <https://miro.com/>.

Spis rysunków

1.1	Logo aplikacji TrafficFlowOptimizer	5
1.2	Eurostat czasy dojazdów	6
1.3	Dysproporcja ruchu drogowego	7
1.4	Wytlumaczenie punktów	8
1.5	Schemat działania aplikacji - klient	14
1.6	Wykorzystane technologie	14
2.1	Ekran tworzenia skrzyżowania	22
2.2	Ekran podglądu skrzyżowania - obraz	23
2.3	Ekran podglądu skrzyżowania - dane	23
2.4	Ekran listy skrzyżowań	24
2.5	Ekran dodawania nagrania	25
2.6	Ekran wykrywania pojazdów	26
2.7	Ekran zlecenia optymalizacji	26
2.8	Ekran wyników - symulacja	27
2.9	Ekran wyników - opisy	27
2.10	Przepływ aplikacji	30
2.11	Lista skrzyżowań	30
2.12	Podgląd skrzyżowania	31
2.13	Przykładowy plik pierwotnej sekwencji świateł	35
2.14	Modal dodawania pierwotnej sekwencji	35
3.1	Architektura projektu	41
3.6	Struktura katalogu src	43
3.7	Przepływ DrawingTool	44
3.8	TFO ciemny tryb	49
3.9	TFO jasny tryb	49
3.10	Architektura Bazy Danych	56
3.11	Schemat Bazy Danych	57
4.1	Wstępny plan aplikacji	65
4.2	Wstępna koncepcja przepływu - Frontend	66
4.3	Szczegółowa koncepcja przepływu - Frontend	66
4.4	Repozytoria - GitHub	69
4.5	Jira w formie Google Sheets	69
5.1	Zrzut ekranu testów Backend	72
5.2	Model skrzyżowania	72
5.3	Nagranie al. Kijowskiej, pierwsza klatka	73

5.4	Nagranie ul. Nawojki, pierwsza klatka	73
5.5	Nagranie ul. Czarnowiejskiej, pierwsza klatka	73
5.6	Zlecenie optymalizacji	74
5.7	Wyniki w formie opisowej	75
5.8	Wyniki w formie symulacji	76

Spis tabel

5.1	Porównanie liczb samochodów	74
5.2	Porównanie stosunków samochodów	75
5.3	Porównanie wariancji stosunków	75