Struktura Aplikacji

Aplikacja Traffic Flow Ninja składa się z dwóch zasadniczych części: obsługa api i model. Są one razem łączone za pomocą kontrolera który jest bezpośrednio używany przez konsolowy interfejs użytkownika.

W poniższym umlu można zobaczyć interfejsy zarówno providera i modelu. W rzeczywistości jest to dynamiczny import pythona, jednak na potrzeby prezentacji graficznej użyliśmy bloczka interfejsu. Funkcjonalność ta pozwala na rozszerzenie aplikacji o inne modele czy providery.

Provider to klasa korzystająca z api i w przypadku naszego providera z narzędzi pomocniczych (geo_utils, polish_roads). Celem providera jest dostarczenie przetworzonych danych na temat wybranych dróg do kontrolera, aby można było ich użyć w połączeniu z modelem.

Obiekty Road i Fragment są zasilane danymi z api a sam model otrzymuje obiekt Road i z niego wyciąga potrzebne informacje aby zwrócić dzienne natężenie ruchu.

Rozwój

W przypadku chęci rozwoju aplikacji o własny provider należy zadbać o:

- Dziedziczenie po interfejsie „RoadProvider”, oraz implementację funkcji get_current_speed, provide i names.

- Napisać bezargumentową funkcję która zainicjalizuje interfejs. I umieścić ją w odpowiednim miejscu w module (__init__.py).

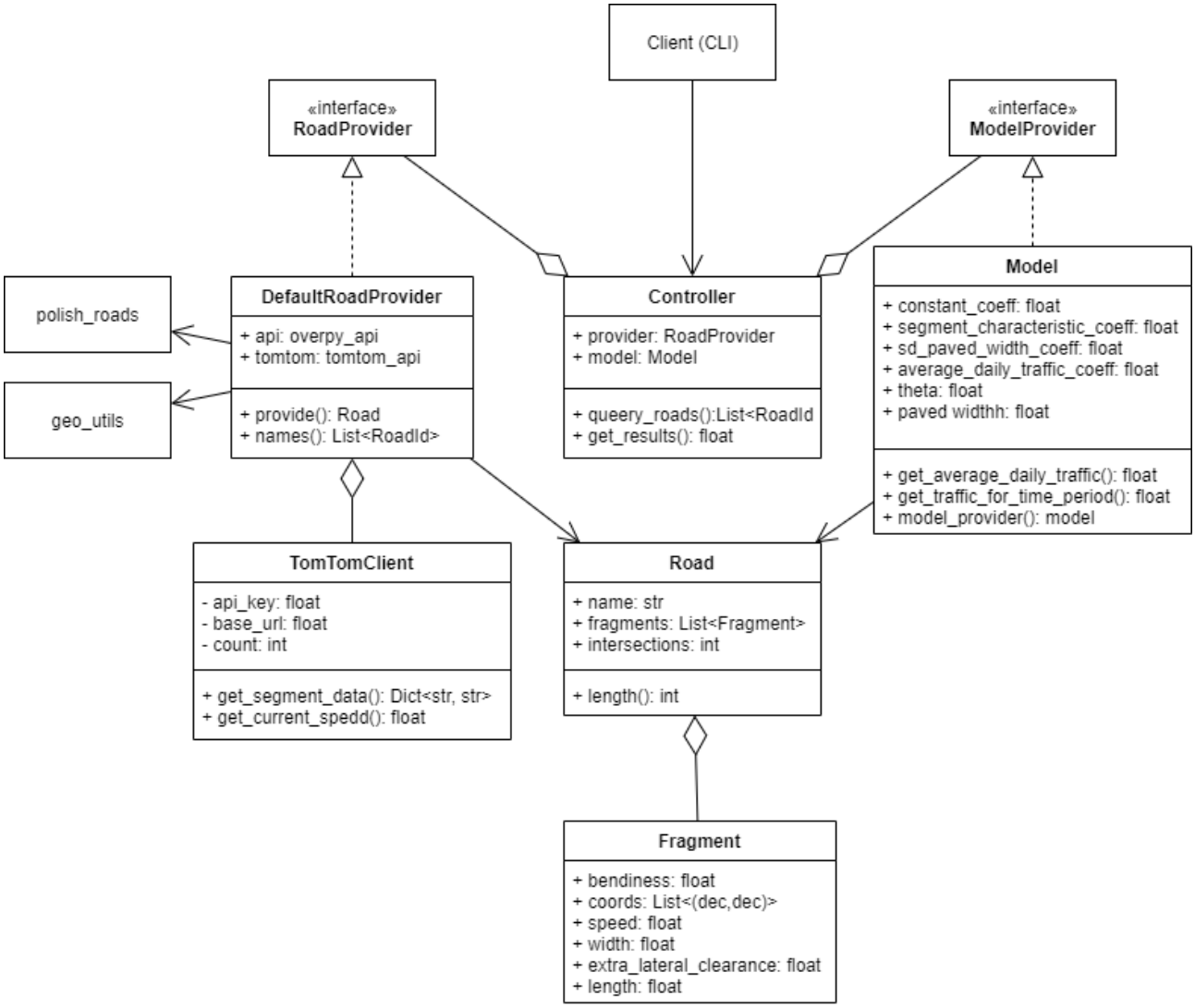
```
def _create_road_provider(tomtom_key: str) -> RoadProvider:
    return DefaultRoadProvider(tomtom_key)
```

```
from default_road_provider.default_road_provider import _create_road_provider
```

- Obsługę przetwarzania informacji z api bądź innych źródeł. Można także wykorzystać nasze rozwiązanie.

- W konstruktorze kontrolera w mainie należy podać nazwę swojego providera.

- funkcja „names” powinna zwracać, po podaniu jej do funkcji „provide”, odpowiadającą drogę i reprezentację tej drogi w postaci zrozumiałej dla człowieka.


Client (CLI)
«interface»
RoadProvider
«interface»
ModelProvider
polish_roads
geo_utils
DefaultRoadProvider
+ api: overpy_api
+ tomtom: tomtom_api
+ provide(): Road
+ names(): List<RoadId>
Controller
+ provider: RoadProvider
+ model: Model
+ queery_roads():List<RoadId
+ get_results(): float
Model
+ constant_coeff: float
+ segment_characteristic_coeff: float
+ sd_paved_width_coeff: float
+ average_daily_traffic_coeff: float
+ theta: float
+ paved widthh: float
+ get_average_daily_traffic(): float
+ get_traffic_for_time_period(): float
+ model_provider(): model
TomTomClient
- api_key: float
- base_url: float
- count: int
+ get_segment_data(): Dict<str, str>
+ get_current_spedd(): float
Road
+ name: str
+ fragments: List<Fragment>
+ intersections: int
+ length(): int
Fragment
+ bendiness: float
+ coords: List<(dec,dec)>
+ speed: float
+ width: float
+ extra_lateral_clearance: float
+ length: float