Nabuchodonozor

Zespół 04

8 maja 2007

Spis treści

Rozdział 1

Wstęp

Projekt *Nabuchodonozor* powstał z myślą o osobach interesujących się wyprawami pieszymi lub rowerowymi zarówno szlakami turystycznymi jak i trasami mniej popularnymi.

Główną częścią projektu *Nabuchodonozor* jest moduł *Semiramida*, czyli portal internetowy, którego zadaniem jest wizualizacja tras użytkowników, przy wsparciu Google Maps. Największą zaletą tego portalu jest powszechny dostęp do tras nieoznaczonych na mapach, jakości ich nawierzchni oraz lokalizacji miejsc wartych zwiedzenia. Ponadto moduł ten umożliwia filtrowanie wyświetlanych tras, wyszczególnianie ich określonych cech, dodawanie komentarzy i zdjęć oraz eksportowanie do różnych formatów.

Dodawanie nowych tras polega na przesłaniu do *Semiramidy* pliku zawierającego listę współrzędnych geograficznych punktów tworzących przebytą trasę. Plik taki można stworzyć przy użyciu modułu *Gilgamesh*, czyli aplikacji Java działającej na telefonie komórkowym, łączącej się z urządzeniem GPS poprzez Bluetooth. Moduł ten pozwala na łatwe korzystanie z urządzenia GPS do rejestrowania tras z dużą dokładnością, określanie jakości nawierzchni oraz oznaczanie położenia miejsc szczególnie przydatnych podczas wyprawy pieszej bądź rowerowej, jak i miejsc zwyczajnie ciekawych i wartych zwiedzenia. Nie mniej przydatną opcją modułu *Gilgamesh* jest wizualizacja aktualnego położenia względem tras pobranych uprzednio z *Semiramidy*. Daje to użytkownikom doskonałą orientację w terenie oraz zabezpiecza przed zwyczajnym zgubieniem się lub zejściem z wyznaczonej trasy.

Innym sposobem dodania do *Semiramidy* nowych tras jest przesłanie pliku stworzonego przez rozbudowane urządzenie GPS, niewymagające aplikacji *Gilgamesh*. Każdy plik zapisany w standardzie NMEA lub GPX zostanie poprawnie dodany do bazy tras.

Połączeniem, przechowywaniem danych i zarządzaniem bazą tras zajmuje się moduł *Hammurabi*. Jego istotą jest upraszczanie podobnych tras tak, by

wyeliminować z nich zbędne punkty oraz zwiększyć wiarygodność i dokładność.

Zarówno *Hammurabi* jak i *Gilgamesh* korzystają z modułu *Marduk* służącego do konwertowania danych pochodzących z urządzenia GPS.

Więcej na temat zasad działania modułów można znaleźć w rozdziale "Dokumentacja techniczna".

Część I

Dokumentacja użytkownika

Rozdział 2

Gilgamesh

2.1 Wymagania

Aby móc w pełni korzystać z aplikacji *Gilgamesh*, telefon komórkowy musi spełniać następujące wymagania:

- obsługa języka Java (wersja MIDP: 2.0, wersja CLDC: 1.1)
- usługa Bluetooth
- 100kB wolnego miejsca na aplikacje

UWAGA: Jeśli telefon nie posiada usługi Bluetooth, działanie aplikacji zamyka się w możliwości wizualizacji mapy bez wskazania aktualnego położenia geograficznego.

2.2 Instalacja aplikacji

Przebieg instalacji aplikacji zależy od modelu telefonu. Należy postępować zgodnie z instrukcjami producenta odnośnie instalacji aplikacji Java.

Przykładowa instalacja na telefonie Sony Ericsson K510i wygląda następująco: po pobraniu na telefon komórkowy plików Gilgamesh.jar i Gilgamesh.jad, należy wejść do katalogu, do którego zostały skopiowane powyższe pliki. W menu podręcznym pojawi się opcja **Instaluj**. Po wybraniu tej opcji należy wskazać folder, do którego aplikacja ma zostać zainstalowana.

(a) Wybór aplikacji

(b) Uruchamianie

(c) Menu główne

Rysunek 2.1: Włączanie Gilgamesha

2.3 Obsługa aplikacji

UWAGA: Przed rozpoczęciem użytkowania aplikacji należy upewnić się, czy usługa Bluetooth w telefonie komórkowym jest włączona (niezbędne do pełnego działania aplikacji).

Po włączeniu aplikacji pojawi się ekran menu głównego. Domyślnie aplikacja nie łączy się automatycznie z urządzeniem GPS, nie rozpoczyna także zapisu i odczytu danych z pliku.

2.3.1 Połączenie z urządzeniem GPS – opcja Urządzenia BT

Aby rozpocząć pobieranie danych o położeniu geograficznym, trzeba utworzyć połączenie z urządzeniem GPS. W tym celu należy włączyć GPS-a oraz Bluetooth w urządzeniu mobilnym, a następnie wybrać opcję **Urządzenia BT**. Po wybraniu tej opcji nastąpi wyszukanie dostępnych poprzez Bluetooth urządzeń oraz ich wylistowanie. Po wybraniu urządzenia GPS, z którego mają być pobierane dane, naciskamy przycisk **Połącz** a następnie zgadzamy się na utworzenie klienckiego połączenia Bluetooth (w przypadku gdy nie odnaleziono od-

powiedniego urządzenia, należy sprawdzić czy na pewno usługa Bluetooth w urządzeniu mobilnym jest aktywna, a odbiornik GPS włączony, następnie powrócić do menu i powtórzyć opisywaną czynność).

(a) Wyszukiwanie urządzeń (b) Dostępne połączenia (c) Połączono z GPS

Rysunek 2.2: Łączenie aplikacji z odbiornikiem GPS

W momencie wybrania opcji **Urządzenia BT**, gdy połączenie jest ustanowione, na ekranie pojawi się komunikat, informujący o aktywnym połączeniu i możliwości rozłączenia.

2.3.2 Rozpoczęcie zapisu danych do pliku – opcja Rozpocznij zapis

Aby rozpocząć rejestrację ścieżki, należy wybrać opcję **Rozpocznij zapis**. Jeżeli aplikacja odnajdzie katalog, do którego ma zapisać dane (domyślnie katalog sounds), na ekranie pojawią się pytania o pozwolenie na odczyt i zapis danych użytkownika - należy wyrazić zgodę. Jeśli katalogu nie uda się odnaleźć (zależne od modelu telefonu), aplikacja poprosi o podanie pełnej ścieżki katalogu, do którego ma zapisywać dane. W przypadku podania nieistniejącego katalogu zostanie ponowiona prośba o podanie ścieżki. Można również w tym momencie zrezygnować z zapisu, wybierając opcję **Menu**. Gdy podana zostanie poprawna

ścieżka, aplikacja zapyta o zezwolenie na odczyt i zapis danych użytkownika, należy wtedy postępować jak wyżej.

Jeśli przed rozpoczęciem rejestracji zostało ustanowione połączenie z urządzeniem GPS, odbierane dane o położeniu geograficznym będą od tego momentu zapisywane do pliku. W przeciwnym wypadku zostanie utworzony pusty plik, a zapis danych do tego pliku nastąpi po połączeniu z urządzeniem GPS i odebraniu pierwszych danych.

UWAGA: Istnieje także możliwość skonfigurowania aplikacji tak, aby powyższe czynności (zapisywanie i łączenie) wykonywały sie automatycznie (patrz **Ustawienia**). Jedyna ingerencja użytkownika polega na wyrażeniu zgody na utworzenie klienckiego połączenia Bluetooth oraz odczyt i zapis. Telefon komórkowy automatycznie połączy się z pierwszym urządzeniem dostępnym poprzez Bluetooth i rozpocznie zapis odczytanych danych do pliku.

2.3.3 Kończenie zapisu danych do pliku – opcja Zakończ zapis

Po rozpoczęciu zapisu w menu głównym zamiast opcji **Rozpocznij zapis** pojawi się **Zakończ zapis**, służące do przerwania rejestracji trasy. By zapobiec przypadkowemu zakończeniu rejestracji, po wybraniu powyższej opcji pojawi się prośba o potwierdzenie.

2.3.4 Pasek informacyjny

W górnej części ekranu **Menu głównego** oraz **Oknie danych GPS** znajduje się pasek informujący o:

- rodzaju nawierzchni (informacja ta nie jest wyświetlana, gdy trasa nie jest rejestrowana)
- stanie połączenia z urządzeniem poprzez Bluetooth (jeśli połączenie jest ustanowione, na pasku znajduje się informacja o nazwie podłączonego urządzenia)
- rejestracji trasy do pliku (jeśli ta opcja jest aktywna, na pasku znajduje się informacja o nazwie pliku)
- ewentualnych błędach podczas tworzenia połączenia lub pliku

2.3.5 Okno danych GPS

Jest to okno prezentujące dane odczytane z GPS-a oraz dodatkowe dane przydatne rowerzyście. Znajdują się w nim następujące informacje:

- Współrzędne geograficzne - szerokość i długość geograficzna
- Wysokość n.p.m.
- Czas UTC
- Przebyta droga wyrażona w metrach
- Czas zapisu – czas, który upłynął od chwili rozpoczęcia rejestracji
- Ilość satelitów – ilość satelitów, z których obecnie GPS pobiera dane
- Dokładność pomiarów wyrażona w decymetrach – maksymalne możliwe odchylenie od prawdziwego położenia, zależne od urządzenia GPS

W zależności od preferencji użytkowania można wyświetlać tylko część tych danych (patrz **Ustawienia**). W zależności od ilości dostępnych satelitów ilość i jakość dostarczanych danych się zmienia. Gdy GPS jest w trakcie wyszukiwania satelitów, dostarcza jedynie informacji o czasie UTC i ilości dostępnych satelitów.

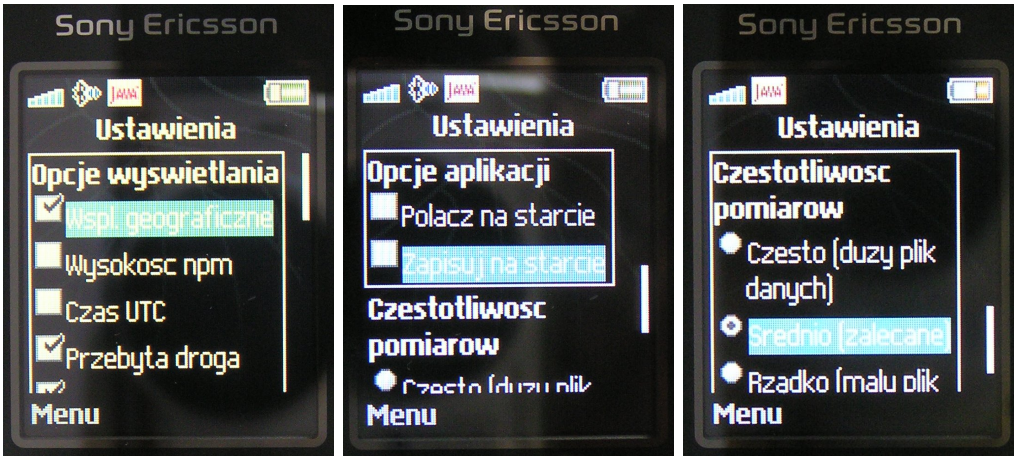
2.3.6 Zmiana opcji aplikacji – Ustawienia

Po wybraniu tej opcji pojawi się ekran z opcjami ustawień aplikacji, podzielonymi na 3 sekcje:

Sekcja 1. Opcje wyświetlania Sekcja ta odnosi się do ekranu danych GPS. Określa, które informacje mają być widoczne, a które nie. Zaznaczenie lub odznaczenie danej opcji odbywa się poprzez wybranie odpowiedniego

elementu i naciśnięcie przycisku **Wybierz** (lub innego zależnego od modelu telefonu). Opis znaczenia poszczególnych opcji znajduje się w rozdziale **Okno danych GPS**.

Sekcja 2. Opcje aplikacji Zaznaczenie opcji **Połącz na starcie** spowoduje, że od razu po włączeniu aplikacji, począwszy od następnego uruchomienia, nastąpi automatyczne połączenie z pierwszym znalezionym urządzeniem. Zaznaczenie opcji **Zapisuj na starcie** spowoduje, że od razu po włączeniu aplikacji, począwszy od następnego uruchomienia, zostanie stworzony plik i zostanie rozpoczęta rejestracja ścieżki.



(a) Sekcja 1 (b) Sekcja 2 (c) Sekcja 3

Rysunek 2.3: Opcje aplikacji

Sekcja 3. Częstotliwość pomiarów W sekcji tej istnieje możliwość wybrania jednej z trzech częstotliwości pomiarów, wpływającej na częstość aktualizowania danych w **Oknie danych GPS** oraz ilość punktów zapisywanych do pliku. W przypadku jazdy szybkim pojazdem zaleca się wybranie opcji **Często**. Gdy telefon komórkowy nie posiada dużo wolnego miejsca, zaleca się wybranie opcji **Rzadko**. Opcję tą można wielokrotnie zmieniać podczas użytkowania aplikacji.

UWAGA: Opcji aplikacji nie trzeba konfigurować przy każdym uruchomieniu komórki, są bowiem zapamiętywane.

2.3.7 Wizualizacja

Po wybraniu w menu głównym opcji **Wizualizacja**, wyświetla się ekran z mapą narysowaną na podstawie pliku, zawierającego paczkę ścieżek. Trasy zaznaczane są kolorami odpowiadającymi rodzajom nawierzchni. Paczka ścieżek domyślnie znajduje się w katalogu sounds. Jeśli lokalny system nie zwróci ścieżki dostępu do tego katalogu, użytkownik zostanie poproszony o jej wprowadzenie (można wprowadzić scieżkę do dowolnego katalogu, w którym znajdują si dane).

Rysunek 2.4: Legenda - rodzaje nawierzchni

Zaznaczane są również zdefiniowane punkty specjalne z poniższymi opisami:

- sklep
- pole – pole namiotowe
- WC
- plaza – plaża, kąpielisko
- kościół

UWAGA: po oddaleniu się (od poziomu 4), punkty specjalne są wyświetlane jedynie jako pomarańczowe kwadraciki w celu zwiększenia przejrzystości mapy.

Jeśli odebrano dane o położeniu geograficznym, mapa centruje się w miejscu, w którym znajduje się użytkownik. Nasze położenie zaznaczone jest czerwonym kwadratem. Aby ułatwić oglądanie mapy, dostępne są następujące funkcje:

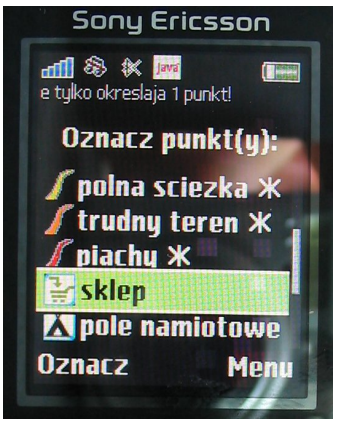
- klawisz 1 – oddalenie się, co za tym idzie zwiększenie obszaru widzianego.
- klawisz 3 – przybliżenie się, zmniejszenie obszaru widzianego.
- klawisze 2, 4, 6, 8 – przesuwanie mapy odpowiednio w górę, lewo, prawo, dół. Aby umożliwić komfortowe oglądanie dalszych terenów, gdy mapa jest przesunięta nie centruje sie w miejscu położenia użytkowania.
- klawisz 5 – centruje mapę w miejscu położenia użytkownika zachowując obecne przybliżenie. Po naciśnięciu tego klawisza mapa po każdorazowym odebraniu danych centruje się w miejscu w jakim znajduje się użytkownik.

UWAGA: Aby zwiększyć przejrzystość, od 4 poziomu oddalenia grubość ścieżek zostaje zmniejszona.

W oknie wizualizacji znajduje się opcja **Oznacz**, służy ona do wstawiania punktów specjalnych oraz punktów zmiany nawierzchni.(Dokładny opis znajduje się w podrozdziale **Oznaczanie punktów**)

2.3.8 Oznaczanie punktów

Podaczas rejestrowania trasy aplikacja *Gilgamesh* daje możliwość oznaczenia jakości nawierzchni, po której się aktualnie jedzie lub idzie. Ponadto istnieje także możliwość zaznaczenia lokalizacji ważnych miejsc, takich jak pole namiotowe, sklep czy kąpielisko oraz własnoręcznego dodania opisu tekstowego do obecnego miejsca. Ułatwia to znacznie innym użytkownikom *Semiramidy* planowanie trasy wycieczki.



Wszystkie te możliwości są dostępne poprzez naciśnięcie klawisza **Oznacz**. Spowoduje to pojawienie się listy predefiniowanych punktów. Pozycje

oznaczone '*' zmieniają jakość nawierzchni wszystkich następujących punktów, począwszy od ostatniego, który został zarejestrowany. Pozycje nieoflagowane '*' oznaczają tylko ostatnio zarejestrowany punkt. Ostatnia pozycja służy do dodawania własnego opisu punktu. W tym przypadku punkt, do którego dodawany jest opis, również jest ostatnim zarejestrownym punktem. Ta zmiana nie dotyczy następujących punktów – jeśli oznaczana była jakość nawierzchni, to nadal będzie ona oznaczana bez potrzeby ponownego ustawiania rodzaju nawierzchni. W przypadku, kiedy żaden punkt nie został jeszcze zarejestrowany, punkty nieoflagowane '*' będą pomijane.

Po wybraniu odpowiedniego opisu punktu i po naciśnięciu klawisza **Oznacz** automatycznie aplikacja powróci do ostatniego ekranu – **Ekranu wizualizacji** lub **Ekranu danych GPS**, bądź przejdzie do ekranu ręcznej edycji opisu. W ostatnim przypadku po wprowadzeniu tekstu i wciśnięciu klawisza **Oznacz**, również nastąpi powrót do ostatnio oglądanego ekranu.

2.4 Zgrywanie zebranych danych na komputer

Po zakończeniu pracy z aplikacją zebrane dane można zgrać na komputer. Odbywa się to zgodnie z zaleceniami producenta telefonu komórkowego. Jeśli ścieżka do danych podczas działania aplikacji nie została zmieniona, zarejestrowane dane znajdują się domyślnie w katalogu z dźwiękami.

2.5 Dodawanie pakietów ścieżek

Aby podczas wizualizacji pojawiły się ścieżki, należy pobrać z serwisu *Semiramnida* pakiet danych. Następnie dane te należy wgrać do folderu z dźwiękami w telefonie komórkowym. Gdy telefon nie posiada takiego katalogu, należy wgrać dane do dowolnego katalogu z możliwością odczytu (Podczas włączania wizualizacji aplikacja poprosi o podanie pełnej ścieżki do tego katalogu).

Rozdział 3

Semiramida

Semiramida jest serwisem internetowym umożliwiającym dodawanie nowych ścieżek, marków wraz ze zdjęciami, oglądanie zawartości bazy oraz jej komentowanie. Serwis korzysta z map Googla, dzięki czemu drogi i wyróżnione punkty można oglądać na tle mapy lub zdjęć satelitarnych, albo jednych i drugich. Serwis daje możliwość stworzenia swojego konta użytkownika.

3.1 Wymagania

Przeglądarka internetowa

Semiramida wspólpracuje z wiekszością przegladarek internetowych, które wspierają współczesne standardy. Dotychczas była sprawdzana na nastepujacych przeglądarkach:

- Firefox 1.5, Firefox 2.0 bezproblemowo dziala
- Opera 9 bezproblemowo dziala
- Konqueror 3.5 bezproblemowo dziala
- Microsoft Internet Explorer 6.0 działa, lecz pojawiają się problemy podczas wyświetlania mapy (wyświetlanie jest bardzo powolne, występują problemy z przezroczystymi grafikami), *polecamy korzystać z innych przegladarek*

Aby móc korzystać z *Semiramidy* należy mieć włączoną obsługię emphJavaScript. Jest ona wymagana podczas pracy z mapą.

Obsługa *ciasteczek* (*cookie*) nie jest wymagana, lecz mimo to warto jest mieć ją właczoną. W ten sposób *Semiramida* może zapamiętać przykładowo jaki obaszar mapy ostatnio oglądaliśmy.

System operacyjny i sprzęt

Zasadniczo *Semiramida* powina pracować dobrze na każdym systemie operacyjnym dla którego istnieje odpowiednia przeglądarka. Konfiguracja sprzętowa może wpłynąć jedynie na szybkość działanie serwisu, lecz sam moduł powinien działać na każdym komputerze, który spełnia wymagania dotyczące oprogramowania.

3.2 Korzystanie z serwisu

Po otwarciu strony *Semiramidy* ukaże się strona główna. Na samej górze strony znajduje się pasek logowania. Jeśli nie jest się zalogowanym na pasku wyświetli sie pytanie czy użytkownik chce sie zalogować czy zarejestrować. Klikając na **zalogować** przechodzi się do okna logowania. W polu *Nazwa użytkownika* wpisać należy swoją nazwę, a w polu *Hasło* hasło. Następnie należy nacisnąć **Zaloguj**. Od tej pory jest się zalogowanym i pasek u góry zmienia wygląd. Klikając na **wylogować** można się wylogować. Jeśli osoba nie ma jeszcze konta należy na pasku wybrać **zarejestrować**. Otworzy się nowa strona, gdzie należy podać nazwę użytkownika, hasło i powtórzyć hasło. Kilkając na **Utwórz konto** zostanie utworzone nowe konto.

Jeśli z dowolnego miejsca serwisu chcemy wrócić na stronę główną wystarczy kliknąć tylko na nazwę **Semiramida** lub logo. Podobnie jeśli chcemy przejść do mapy wystarczy tylko nacisnąć na ikonkę kuli ziemskiej lub słowo **Mapa**.

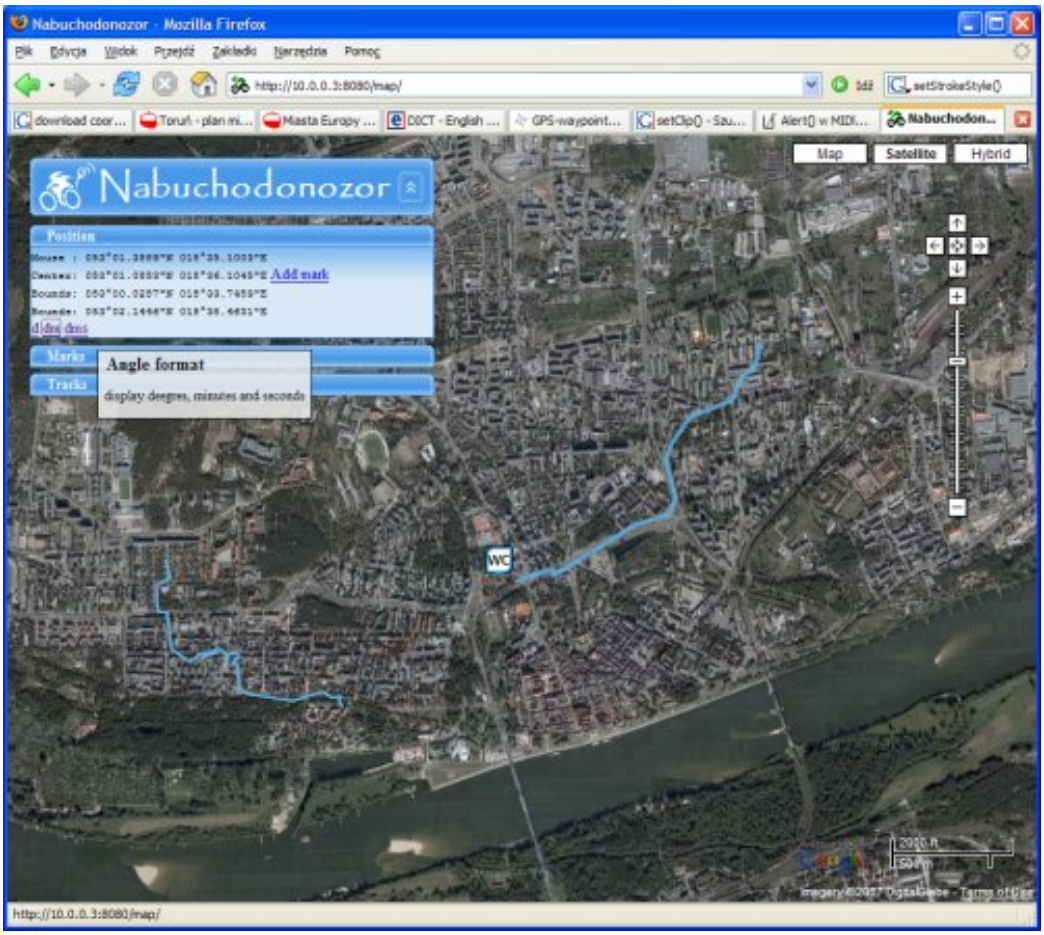
Poniżej loga i nazwy serwisu na wszystkich stronach oprócz mapy jest pasek menu. Poszczególne pozycje zostaną omówione poniżej. Po prawej stronie paska znajdują się dwie flagi. Klikając na flagę polską język serwisu zmieni się na polski, klikając na flagę brytyjską zostanie ustawiony język angielski.

3.2.1 Strona główna

W lewej, górnej części strony głównej znajduje się dziesięć ostatnio dodanych ścieżek. Poniżej wyświetlanych jest dziesięć ostatnio dodanych znaczników, a jeszcze niżej zdjęć. Każdy z wyświetlanych elementów jest odnośnikiem. Po klikniecu na niego pojawia się strona opisująca ten obiekt. Opis ścieżki i znacznika zostanie omówiony w rozdziale *Ścieżki* oraz *Znaczniki*. W prawej części strony głównej znajduje się opis serwisu i modułu *Gilgamesh*. Klikając na ikonkę telefonu komórkowego lub link **Pobierz Gilgamesha** można ściągnąć wyżej wymienioną aplikację.

3.2.2 Mapa

Po najechaniu na dowolny element pokazuje się dymek opisujący dany element.



Rysunek 3.1: Semiramida wyświetlająca współrzędne

Logo

Z mapy korzysta się używając menu po lewo. Naciskając na górny pasek (słowo **Semiramida** lub logo) nastąpi powrót na stronę główną. Klikając na strzałki po prawej stronie paska można zminimalizować całe menu. Tym samym przyciskiem przywracamy je do poprzedniej wielkości.

Pozycja

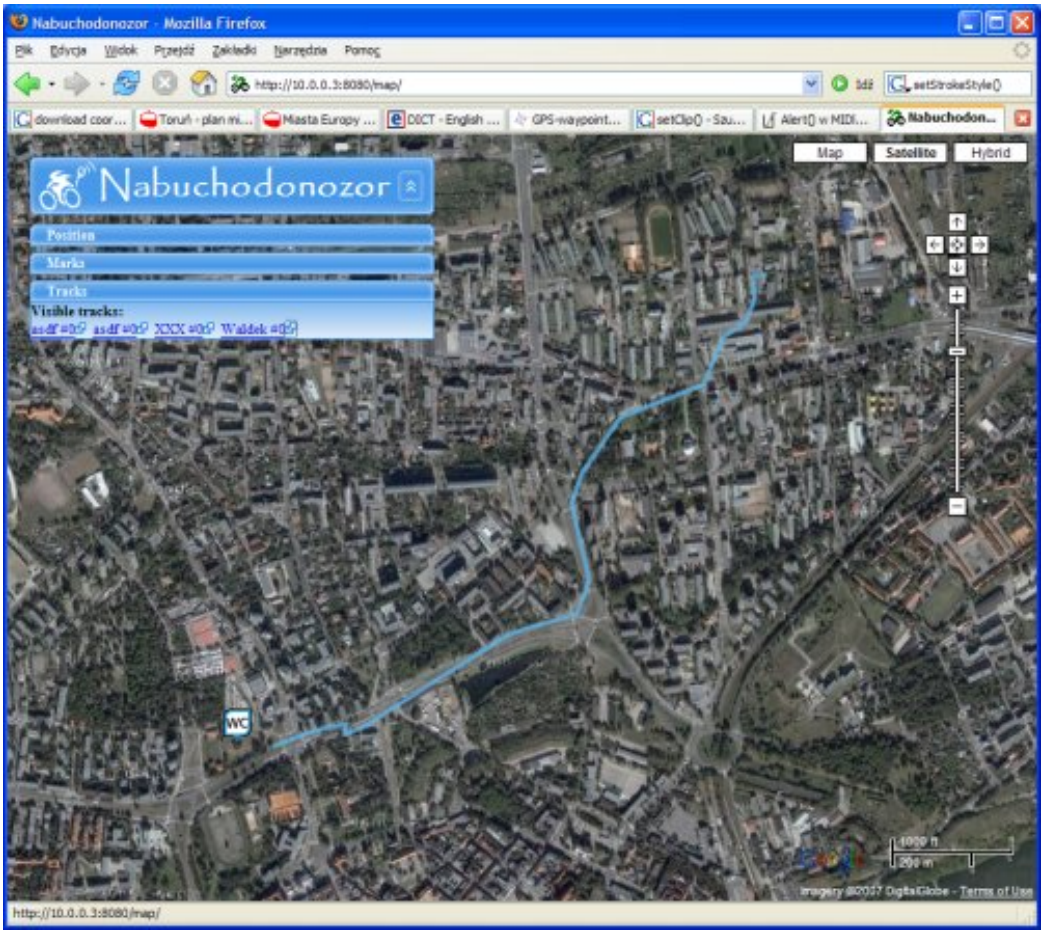
Wybierając pasek **Pozycja** rozwija się pole w którym znajdują się (od góry) współrzędne punktu, gdzie znajduje się kursor, współrzędne środka wyświetlanego obszaru oraz współrzędne wyświetlanego obszaru. Na dole można zmienić format wyświetlania, po klikniecu na pierwszy przycisk położenie opisywane jest w stopniach, na drugi w stopniach i minutach, na trzeci w stopniach, minutach i sekundach. W rzędzie zawierającym punktu środka wyświetlanego obszaru znajduje się *Dodaj znacznik*. Po kliknięciu w ten link otwiera się okno dodawana nowego marku. Ma on ustawione współrzędne środka obszaru, możemy tu zmieniać rodzaj marka, jego nazwę i opis. Przycisk **Dodaj znacznik** dodaje punkt do bazy.

Ścieżki

Wybierając pasek **Ścieżki** wyświetlone zostaną ścieżki z widzianego obszaru. Klikniecie na nazwę trasy powoduje przekierowanie na stronę z informacjami o ścieżce (dział *Ścieżki*). Wybierając strzałkę obok nazwy trasa stanie się pogrubiona i mapa wycentruje się na nią. Odznaczenie którejś ze ścieżek spowoduje, że ścieżka ta nie będzie widoczna. Po ponownym zaznaczeniu zostanie ona znowu wyświetlona. U góry sekcji znajdują się opcje wyświetlania ścieżek. Są to w kolejności od lewej: **Id** - ścieżki wyświetlane są w kolorze nadanym przez użytkownika, **Kind** - ścieżki wyświetlane są w kolorach odpowiadającym ich typowi nawierzchni, **Random** - kolor jest losowy, **One** - wszystkie trasy są w jednym kolorze. **Visible All** zaznacza wszytskie ścieżki, a **Visible None** odznacza je.

Znaczniki

Po kliknięciu na pasek **Znaczniki** wyświetlą się wszystkie aktulanie widziane marki. Naciskając na nazwę marku wyświetlana jest strona zawierająca o nim informacje (podobnie jak na stronie głównej, omówione w sekcji *Znaczniki*). Klikając na strzałkę obok nazwy ekran zostanie wycentrowany na dany punkt specjalny. U góry znajdują się ikonki reprezentujące rodzaje znaczników. W kolejności od lewej są to: marki z dodatkowym opisem, sklepy, pola namiotowe, toalety, kąpieliska, kościoły. Klikniecie na ikonę powoduję, że znaczniki danego typu nie będą wyświetlane, a obrazek stanie się szary. Ponowne wybranie tego samego elementu spowoduje, że marki danego typu znowu zostaną pokazane na mapie.



Rysunek 3.2: Wybieranie ścieżek

Legenda

W legendzie znajdują się rodzaje tras oraz odpowiadające im kolory dróg na mapie.

Eksport

Dane, które aktualnie widać na mapie można wyeksportować do któregoś z dostępnych formatów. Klikając na ikonę lub nazwę typu uzyskamy plik w formacie (od lewej): kml, pdf, ps, gpx oraz txt (pakiet ścieżek do wizualizacji w aplikacji *Gilgamesh*.

Obszar mapy

Wyświetlana mapa na funkcjonalności Google Maps. Po prawo znajdują się strzałki do nawigacji. Poniżej pasek na którym ustawia się przybliżenie. U samej góry mamy do wyboru rodzaj mapy - zwykła, zdjęcia satelitarne oraz połączenie obu rodzajów. Jeśli na obszarze mapy kliknie się na znaczki pokaże się dymek zawierający jego opis oraz link to strony z danymi marka.

3.2.3 Ścieżki

Wybierając z górnego menu pozycje **Ścieżki** wyświetlą się wszytskie ścieżki jakie są w bazie danych. Ścieżki wyświetlane są po dziesięć na stronie. Jeśli ścieżek jest mniej, wyświtlane są wszytskie ścieżki. Jeśli jest ich więcej kolejne ścieżki znajdować się będą na kolejnych stronach. U góry znajduje się menu nawigacji. Wybierając **<<pierwsza** przeniesiemy się na pierwszą stronę ze ścieżkami, **ostatnia>>** na stronę ostatnią, przyciski **<poprzednia** i **następna>** wyświetlą poprzednią i następną stronę w stosunku do obecnej. Zawsze też można klikąć na numer strony.

W oknie wyświetlane są nazwy ścieżek, informacje kiedy została dana ścieżka dodana, przez kogo oraz jej opis. Wybierając nazwę ścieżki zobaczymy stronę z informacjami o niej oraz dodanymi komentarzami. Jeśli użytkwnik jest zalogowany może dodać komentarz. Tekst wpisuje się w oknie u dołu, następnie należy nacisnąć **Dodaj komentarz**. Właściciel lub moderator może również edytować lub usunąć ścieżkę z bazy. W oknie edycji można zmienić nazwę, opis lub kolor ścieżki. Następnie należy zapisać zmiany naciskając **Zapisz zmiany**. UWAGA: Ze względów bezpieczeństwa znaczniki HTMLa są usuwane. Można jednak używać przyjaznego dla użytkownika języka Markdown, który daje porównywalne możliwości, jednocześnie będąc prostszym i bardziej intuicyjnym od HTMLa. To samo dotyczy komentarzy do Znaczników (patrz sekcja *Znaczniki*). Jeśli użytkownik jest autorem komentarza lub moderatorem może go edytować naciskając **Edytuj**. W oknie, które się wyświetli, należy zmienić komentarz i nacisnąć **Zapisz zmiany**. Jeśli nie chcemy ich zapisywać należy wybrać **Porzuć zmiany**. Jeśli użytkonik może edytować komentarz, może też go usunąć. Po wybraniu **Usuń** pojawi się pytanie czy na pewno chce się usunąć komentarz. Wybranie **Usuń!** usunie go ostatecznie. Poniżej przycisków **Edytuj** i **Usuń** wymienione są formaty do których można wyeksportować daną ścieżkę. Klikając na którąś z nazw formatu uzyskamy plik wybranego typu zawierający trasę.

3.2.4 Znaczniki

Okno ze znacznikami wygląda bardzo podobnie jak okno Ścieżek (patrz *Ścieżki*). Znaczniki wyświetlane są po dziesięć na stronie, u góry znajdują się przyciski do nawigacji oraz numery stron z markami. Wybierając nazwę znacznika otworzy się podobne okno jak w przypadku ścieżek. U góry znajduje się opis znacznika, niżej, jeśli takie isnieją, wyświetlane są zdjęcia dodane do marka, potem znajdują się komentarze. Jeśli użytkownik jest zalogowany może on dodać komantarz. Właściciel lub moderator może również edytować lub usuwać komentarze lub marki. W oknie edycji znacznika można zmienić jego współrzędne, typ, nazwę oraz opis. Naciśniecie **Zapisz zmiany** zapisze modyfikacje w bazie. Jeśli do marka dodane jest zdjęcie to klikając na nie można zobaczyć je w powiększeniu. Poniżej znajduje się jego opis oraz data dodania. Klikając na **Usuń** lub **Edytuj** można odpowiednio usunąć zdjęcie lub edytować jego atrybuty. Użytkownik musi mieć oczywiście odpowiednie uprawnienia, ma je właściciel lub moderator. Wybierając **Ściągnij orginał** możemy pobrać plik.

Kolejną opcją jest dodawanie zdjęć. Ta sekcja znajduje się na dole strony. W pierwszym polu należy wpisać opis, niżej ścieżkę do pliku lub używając przycisku **Przeglądaj...** wskazać zdjęcie. Po naciśnieciu **Prześlij zdjęcie** obrazek zostanie dodany.

3.2.5 Prześlij

Ta strona służy do dodawania nowych ścieżek. Można to zrobić przesyłając plik uzyskany za pomocą *Gilgamesha*, plik w formacie gpx lub standardzie NMEA. Kilka plików może być spakowanych do archiwum zip. Aby dodać ścieżkę trzeba być zalogowanym, jeśli użytkownik nie jest zalogowany następuje automatyczne przekierowanie do strony logowania. W pierwszym polu należy wpisać nazwę trasy, poniżej jej opis, a następnie ścieżkę do pliku z danymi lub wskazać ją poprzez przycisk **Przeglądaj...**. Naciśnięcie **Prześlij** doda punkty do bazy danych.

Część II

Dokumentacja techniczna

Rozdział 4

Gilgamesh

4.1 Połączenie z GPS-em

4.1.1 Opis klas

Łączenie telefonu komórkowego z GPS-em odbywa się poprzez Bluetooth. Klasy odpowiedzialne za obsługę tego połączenia to:

- *MyDiscoveryListener*
- *BluetoothDevice*
- *GpsInfo*

GpsInfo to pomocnicza klasa przechowująca adres i nazwę urządzenia.

MyDiscoveryListener (implementujący *DiscoveryListener*) odpowiada za wyszukanie i stworzenie wektora dostępnych urządzeń (elementy typu *GpsInfo*).

W celu uzyskania takiego wektora, za pomocą klasy *MyDiscoveryListener*, należy w pierwszej kolejności stworzyć obiekt tej klasy i wywołać na nim metodę *initialize()*, a następnie *findDevices()*, w celu rozpoczęcia wyszukiwania. Od tego momentu aż do zakończenia odnajdywania dostępnych urządzeń metoda *searchComplete()* będzie zwracać wartość *false*. Po uzyskaniu wartości *true*, wynikiem wywołania metody *getDevices()* będzie żądany wektor. (przykładowe wykorzystanie: klasa *BluetoothDevice*, metoda *searchDevices()*).

BluetoothDevice odpowiada za połączenie i odczytanie danych z urządzenia. Wykorzystuje obiekt klasy *MyDiscoveryListener* w celu zdobycia adresu.

4.1.2 Możliwość rozbudowy aplikacji o inne typy połączeń

Aplikacja *Gilgamesh* zawiera klasę *BluetoothDevice*, implementującą interfejs *Device*. Aby dodać możliwość połączenia z urządzeniem GPS, używającym innego typu komunikacji niż bluetooth (np. USB), należy:

- stworzyć klasę implementującą interfejs *Device*
- zaimplementować wszystkie metody odpowiednio do wybranego typu połączenia
- w klasie *Engine* zmienić typ zmiennej device na nowo powstałą klasę

UWAGA: od tej pory nieużywane będą klasy: *BluetoothDevice* i *MyDiscoveryListener*

Tak zmieniony i skompilowany projekt jest juz gotowy do działania z urządzeniem GPS, używającym innego typu połączenia niż bluetooth.

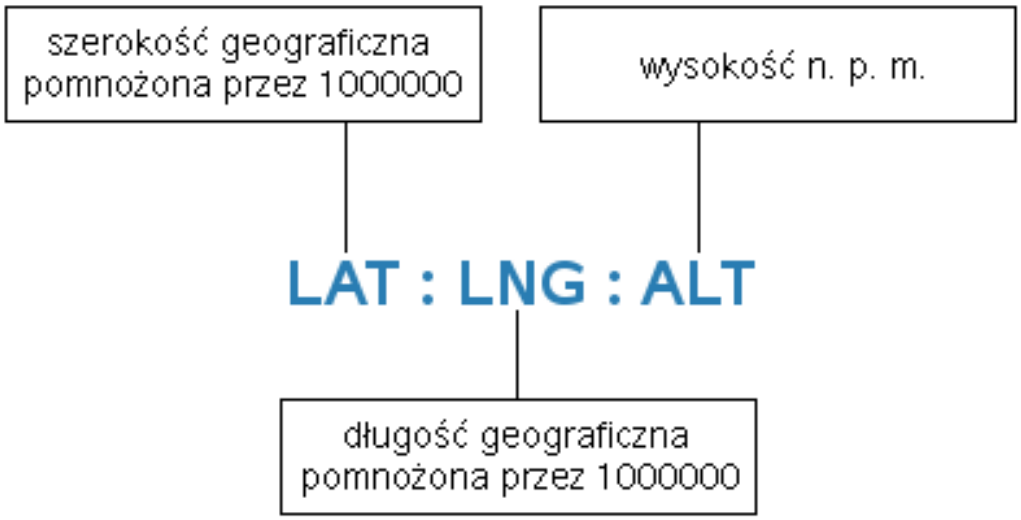
4.2 Przechowywanie danych (zapis, odczyt, format)

4.2.1 Zapis danych pobranych z GPS, format pliku (klasa: ***FileEditor***)

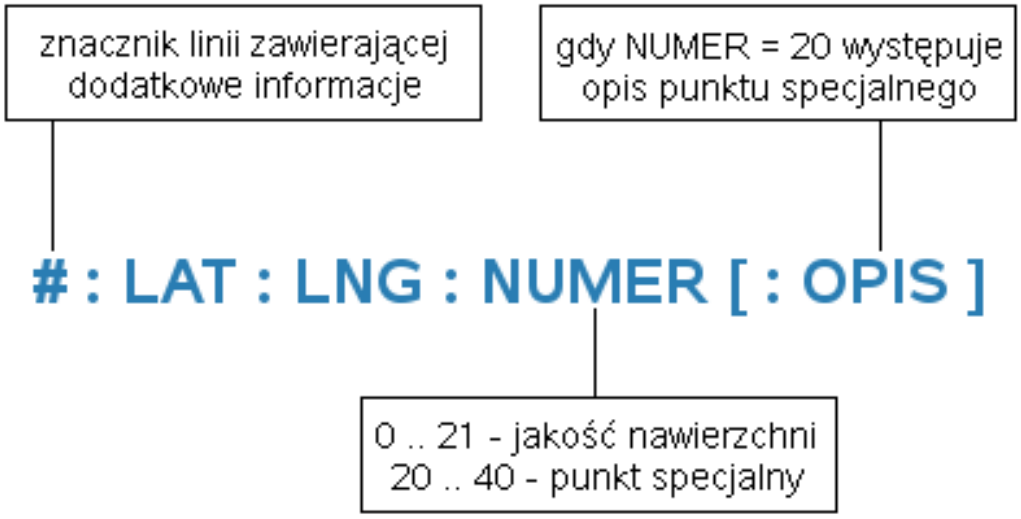
Głównym zadaniem aplikacji *Gilgamesh* jest rejestrowanie danych pobranych z GPS oraz punktów specjalnych i wprowadzonych przez użytkownika rodzajów nawierzchni. Za zapis danych w pamięci telefonu komórkowego odpowiedzialna jest klasa *FileEditor*. W celu ułatwienia użytkownikowi przeglądania stworzonych plików nazwy zawierają pobraną z systemu datę. Format tworzonego pliku jest następujący: w pierwszej linii znajduje się znacznik <Nabuchodonozor>. Następne linie mogą mieć jeden z poniższych formatów:

4.2.2 Odczyt plików ze ścieżkami, format pliku (klasa: ***MyFileReader***)

Klasa *MyFileReader* odpowiedzialna jest za czytanie plików zawierających pakiety ścieżek pobrane z serwisu *Semiramida* (wchodzącego w skład projektu). Format czytanego pliku jest następujący:



(a) Zwykły punkt



(b) Punkt specjalny (zdefiniowane numery: 0 – brak oznaczenia, 1 – ruchliwa szosa, 2 – spokojna szosa, 3 – ścieżka asfaltowa, 4 – ubita droga, żużel, 5 – polna sciezka, 6 – trudny teren, 7 – piachy, 21 – sklep, 22 – pole namiotowe, 23 – WC, 24 – plaża, kąpielisko, 25 – kościół)

Rysunek 4.1: Sposoby zapisu punktów w pliku

```
ścieżka
&ścieżka
&ścieżka
(...)
&ścieżka
```

Format ścieżki jest analogiczny do zapisywanych poprzez aplikację *Gilgamesh* (patrz rys. ...) za wyjątkiem pominięcia zbędnych informacji o wysokości n. p. m.

4.2.3 Przechowywanie opcji aplikacji (klasa: *Database*)

Do trwałego przechowywania ustawień wykorzystywany jest RMS (Record Management System). Jego obsługą zajmuje się klasa *Database*. Zapisywane są osobno zestawy ustawień dotyczących wyświetlanych danych GPS, zachowania aplikacji podczas uruchomienia oraz częstotliwości zapisu danych GPS. Podczas działania aplikacji wszystkie informacje o ustawieniach przechowywane są w lokalnych zmiennych klasy *Database*, odpowiednio: *DispOptions*, *AppOptions*, *FreqOptions*.

Podczas tworzenia obiektu tej klasy – w metodzie *OpenSettings()* – wszystkie ustawienia są wczytywane do wyżej wymienionych zmiennych, o ile baza istnieje. W przeciwnym przypadku, w wywołaniu metody *initializeSettings()*, baza zostaje utworzona a zmienne wyzerowane. Odczyt dowolnej danej z RMS polega na odczytaniu z bazy rekordu o podanym identyfikatorze. Ustawienia wyświetlanych danych GPS mają identyfikatory kolejno od 1 do 7, opcje aplikacji – od 8 do 9 a częstotliwość zapisu położenia – 10. W klasie *Database* przechowywane są jedynie identyfikatory pierwszej danej z zestawu. Dla zestawu *DispOptions* jest to *DispOpID* = 1, dla *AppOptions* - *AppOpID* = 8 a dla *FreqOptions* - *FreqOpID* = 10. Należy pamiętać, że każda pojedyncza opcja wymaga osobnego rekordu w RMS. Zatem, aby dodać nowy zestaw opcji do bazy, należy nadać mu identyfikator większy niż największy istniejący identyfikator. Dla przykładu, chcąc dodać zestaw 3 nowych opcji należy nadać im identyfikator początkowy równy 11 oraz zarezerwować kolejne 2 numery dla pozostałych 2 opcji. Należy także zadbać o to, by podczas tworzenia bazy rekordy dla tych opcji zostały utworzone – w tym celu należy poprawić metodę *initializeSettings()*.

W trakcie działania aplikacji pobranie zestawu opcji z klasy *Database* polega na wywołaniu odpowiedniej metody *getSettings()*, *getAppOptions()* lub *getFreqOptions()*. Sama baza pozostaje w tym czasie zamknięta. W przypadku zapisania nowych ustawień, poprzez wywołanie metod *setDispOptions()*,

setAppOptions() i *setFreqOptions()*, baza zostaje otwarta, odpowiednie rekordy nadpisane i z powrotem baza zostaje zamknięta.

4.3 Wizualizacja (klasy: *Visualization*, *AngleMap*)

Aplikacja mobilna posiada możliwość wizualizowania ścieżek pobranych z serwisu *Semiramida*. Szczegółowy opis formatu plików znajduje się w rozdziale "Odczyt plików ze ścieżkami, format pliku (klasa: *MyFileReader*)". Klasa *Visualization* na początku pobiera zawartość pliku i konwertuje ją na dwuwymiarową tablice ścieżek *paths*, tablice punktów specjalnych *specialPoint* i tablice typów tych punktow *specialPointsType*. Tworzy również tablicę *pavingPointsType*, przechowującą dla każdej ścieżki numery punktów w których następuje zmiana nawierzchni oraz rodzaj nowej nawierzchni(numer). Obie tablice zawierają obiekty typu *AngleMap*. Klasa *AngleMap* przechowuje współrzędną geograficzną, daje możliwość dodania milisekund oraz przyciągania punktu do siatki o zadanej ziarnistości (*moveToMesh(int m)*). Ostatnia metoda jest wykorzystywana do przeliczania współrzędnych w celu narysowania ścieżek na zadanym przybliżeniu. Klasa umożliwia zmiany minimalnego i maksymalnego przybliżenia (zmiana *MIN_ZOOM*, *MAX_ZOOM*).

UWAGA: W celu uzyskania dokładniejszego przybliżenia niż 125 milisekund na piksel należy zmienić metodę *keyPressed()* tak aby kolejne dzielenie 125 przez 2 i mnożenie przez 2 dawało nadal 125.

Rozdział 5

Hammurabi

Moduł *Hammurabi* pośredniczy w kontakcie z bazą danych. Jest to serwer, który przyjmuje zapytania dotyczące bazy danych od modułu *Semiramida* i wykonuje odpowiednie operacje w bazie danych. Moduł ten powstał, ponieważ zwykłe wstawianie danych i ich wydobywanie byłoby nieefektywne. Baza jest przygotowana na przechowywanie ogromnych ilości ścieżek, z których każda może składać z wielu punktów. Dlatego potrzebny jest moduł, który zajmie się usuwaniem zbędnych punktów. Ze względu na możliwość pomyłek odczytów GPS-a, potrzebne jest również wyrównywanie nieregularnych odcinków ścieżek. Istnieje również możliwość dodawania tych samych tras, potrzebne jest więc również uśrednianie kilku ścieżek, które są tożsame, ale ze względu np. na kierunek jazdy lub błędy GPS-a mogą składać się z nieco innych punktów. *Hammurabi* może pobierać dane w kilku formatach oraz kieruje eksportem ścieżek do różnych formatów.

5.1 Plik konfiguracyjny

Moduł *Hammurabi* parametry zczytuje z pliku *conf.txt*, który znajduje się w foldrze projektu. Są tam informacje dotyczące głównie bazy danych. Plik ma postać: "parametr: wartość" i natstępujący wygląd:

```
rodzaj_bazy: typ_baza     // tu nalezy wspisac rodzaj bazy
                          // danych, mysql lub postgres albo
                          // wlasny - patrz nizej
ip:          xx.xx.xx.xx  // adres ip hosta, na ktorym dziala baza
baza:        nazwa_bazy   // nazwa bazy danych
uzytkownik:  uzytkownik   // nazwa uzytkownika bazodanowego
haslo:       haslo        // haslo podanego wyzej uzytkownika
inny:        nr           // rodzaj marka z opisem
podzial:     nr           // ostatni numer oznaczajacy rodzaj drogi
```

5.2 Baza danych

Moduł *Hammurabi* może współpracować z wieloma bazami danych. Za kontakt z bazą odpowiada klasa *Base*. Jest to klasa abstrakcyjna, zawierająca wszystkie metody do wstawiania i pobierania danych, które dziedziczą jej potomkowie. W potomkach tej klasy dopisany został jedynie konstruktor, który łączy się z bazą odpwiedniego typu. Nasz projekt może współpracować z dwoma typami bazy: MySQL (testowane z wersją 5.2) oraz PostgreSQL (testowane z wersją 8.2.3). Do obsługi bazy MySQLowej używana jest klasa *MysqlBase*, a Postgresowej *PostgresBase*. Bardzo łatwo można rozszerzyć *Hammurabiego* o współprace z innymi bazami danych. Należy dopisać jedynie nową klasę, która będzie dziedziczyć z klasy *Base*. W klasie tej należy stworzyć metody odpowidzialne za połączenie z bazą. Następnie w klasie *BaseThread*, w konstruktorze trzeba zmienić sekcję *if* tak, aby po sczytaniu z pliku konfiguracyjnego nowego rodzaju bazy, został stworzony obiekt właściwej klasy. Klasa *Base* używa standardu języka SQL, nie gwarantujemy działania modułu, jeśli baza danych nie będzie zgodna ze standardem.

My używaliśmy bazy Postgres w wersji 8.2.3. Poniżej opisane jest jak taką bazę danych zainstalować i jakie są jej wymagania.

5.2.1 Instalacja bazy danych

- Należy pobrać źródła ze strony http://www.postgresql.org/
- Rozpakować archiwum odpowiednimi poleceniami
- Zainstalować komendami:

  ```
  ./configure
  make
  make install
  ```

- Wygodnie jest dodać ścieżkę do katalogu, w którym zainstalowany jest Postgres do zmiennej PATH. Jeśli się tego nie zrobi, to do poleceń w dalszej części będzie trzeba dodać ścieżkę do katalogu bin w katalogu Postgresa.
- Następnie należy utworzyć katalog, w którym będzie znajdować się nasza baza danych, właścicielem tego katalogu powinien być użytkownik postgres. Z poziomu tego użytkownika należy utworzyć bazę danych poleceniem:

  ```
  initdb -D sciezka_do_utworzonego_katalogu
  ```

- Teraz można już wystartować serwer. Robi się to przy pomocy polecenia:

  ```
  pg_ctl start [opcje]
  ```

- Jeśli nasz serwer bazy danych ma obsługiwać nie tylko lokalne zgłoszenia, należy również zmienić pliki konfiguracyjne: postgresql.conf i pg_hba.conf. W pierwszym pliku należy zmienić linie:

  ```
  # katalog bazodanowy
  data_directory = 'sciezka do katalogu'
  # w ten sposob nasz serwer bedzie nasluchiwal na wszystkich
  # interfejsach maszyny. Jesli chcemy wybrac tylko jeden,
  # to miedzy znaki '' nalezy wpisac adres IP:
  listen_addresses = '*'
  # jesli chcemy zmienic domyslny port, nalezy go ustawic
  # w tej opcji:
  port = nr_portu
  ```

  W pliku pg_hba.conf należy dodać linie postaci:

  ```
  host nazwa_bazy_danych uzytkownik_bazodanowy adres_IP
  maska metoda_autentykacji
  ```

  Jest jeszcze kilka innych rodzajów wpisów, wszystkie są bardzo dobrze opisane w początkowej części pliku.

5.2.2 Wymagania bazy danych

- GNU make
- Kompilator ISO/ANSI C. Rekomendowany jest gcc.
- gzip do odpakowania paczki.
- Biblioteki GNU Readline.
- W czasie kompilacji wymagane jest 65MB na dysku, katalog po zainstalowaniu zajmuje 15MB. Pusty folder bazodanowy zajmuje 25MB.

5.2.3 Tabele bazy danych

Z punktu widzenia modułu *Hammurabi* najważniejsze w bazie danych są tabele *points* i *rect*.

Tabela *points* przechowuje wszystkie punkty ścieżek. Zawiera ona następujące pola:

Pole	Klucz	Typ	Opis
pid	Klucz główny, klucz obcy	int	Numer ścieżki, do której należy punkt.
point_id	Klucz główny	int	Numer punktu w ścieżce.
lat		double	Szerokość geograficzna punktu.
lng		double	Długość geograficzna punktu.
height		double	Wysokość na jakiej leży punkt.
type		int	Rodzaj nawierzchni.

Tabela *rect* przechowuje prostokąty, w których leżą kolejne ścieżki. Zawiera ona następujące pola:

Pole	Klucz	Typ	Opis
pid	Klucz główny	int	Numer ścieżki.
latA		double	Minimalna szerokość geograficzna prostokąta, w którym leży ścieżka.
lngA		double	Minimalna długość geograficzna prostokąta, w którym leży ścieżka.
latB		double	Maksymalna szerokość geograficzna prostokąta, w którym leży ścieżka.
lngB		double	Maksymalna długość geograficzna prostokąta, w którym leży ścieżka.

Moduł bazodanowy wykorzystuje również tabelę *names*, która przechowuje tłumaczenia rodzajów punktów w postaci numerycznej na opis tekstowy. Zawiera ona pola:

Pole	Klucz	Typ	Opis
id	Klucz główny	int	Numer oznaczający rodzaj punktu.
type		text	Słowny opis rodzaju punktu.

5.3 Współpraca z *Semiramidą*

Hammurabi łączy się z bazą danych, ale jest również serwerem, opartym na wątkach. Każde zgłoszenie obsługiwane jest przez nowy obiekt klasy *BaseThread*, który rozszerza *Thread*. Na zgłoszenia od *Semiramidy* serwer

nasłuchuje domyślnie na porcie 1313. Komunikacja między tymi dwoma modułami odbywa się za pomocą pisania do gniazd. Ustalony został protokół komunikacji, który zostanie opisany poniżej. W fazie tworzenia obu modułów chcieliśmy zaimplementować komunikację opartą na protokole XML-RPC, ale testy wykazały, że metoda ta działała bardzo wolno. Aby zyskać na wydajności serwisu zdecydowaliśmy się więc na zwykłe *Sockety*.

Najważniejszym zadaniem modułu *Hammurabi* jest zwracanie ścieżek znajdujących się w podanym prostokącie. Prostokąt jest wyznaczony przez dwa punkty, A i B, gdzie A to punkt o najmniejszej wartości, a B o największej wartości obu współrzędnych. Pierwsza współrzędną jest lat, czyli szerokość geograficzna, a drugą lng, czyli długość geograficzna. Zapytanie o trasy w danym prostokącie zaczyna się od linii zawierającej słowo "track". Następnie są linie postaci: "latA:szerokość geograficzna punktu A", "lngA:długość geograficzna punktu A", "latB: szerokość geograficzna punktu B", "lngB:długość geograficzna punktu B". W przypadku eksportu danych do formatów innych niż wewnętrzny *Semiramidy* istnieje również metoda zwracania ścieżek o podanym numerze. Wpis w takim przypadku ma postać: "id: numer numer numer ...". Jeśli ma być zwrócona tylko jedna ścieżka - podany jest tylko jeden identyfikator, wpisy wyznaczające zwracany prostokąt są niepotrzebne i ignorowane. Zwracany obszar jest ustawiany na obszar, w którym znajduje się ścieżka. Następujące wpisy do gniazda są opcjonalne. Linia postaci "except:id1 [id2, ...]" zawiera identyfikatory ścieżek, które nie mają być zwracane. Jeśli brak tego wpisu zwrócone zostaną wszystkie trasy. Wpis "format:typ" określa typ zwracanego wyniku. Obsługiwane są PS, PDF, KML, GPX, TXT (typ danych tworzonych przez *Gilgamesha*). Pliki przesyłane są w postaci tablicy bajtów. Jeśli nie ma tego wpisu wynik jest w postaci Stringów obsługiwanych przez *Semiramidę*. W takim przypadku każda linia wyniku ma postać "id_scieżki:id_rodzaju_nawierzchni:szerokość_pierwszego punktu długość_pierwszego_punktu szerokość_drugiego_punktu długość_drugiego punktu..." Po otrzymaniu takiego zgłoszenia *Hammurabi* wydobywa odpowiednie dane z bazy danych. Zwracane trasy są wygładzone, a potem przetwarzane do odpowiedniego formatu. Do eksportu do plików wymagane są również marki z tablicy *semiramis_mark*, które leżą w zadanym obszarze oraz zawartość tabeli *names*. Kolejność wszystkich linii oprócz pierwszej jest dowolna.

Dane przesyłane do wstawienia do bazy poprzedza linia postaci "upload". Następnie z gniazda czytany jest plik. *Semiramida* nie wnika w strukturę tego pliku. *Hammurabi* rozpoznaje, jaki jest typ przesyłanych danych. Obsługiwane są pliki formatu GPX, zwrócone przez GPS w standardzie NMEA oraz pliki stworzone przez moduł *Gilgamesh*. Dane są parsowane i wydobywane zostają z nich współrzędne punktów. Następnie nowa ścieżka jest upraszczana i uśredniana z innymi trasami znajdującym się już w tabelach,

które są do niej podobne. Potem punkty są wstawiane do bazy. Jeśli wśród danych znajdują się marki, to moduł przegląda tablicę *semiramis_mark* i sprawdza czy dane marki nie znajdują się już w bazie. Jeśli tak marki powtarzające się marki nie są dodawane. W razie błędu do *Semiramidy* przesyłany jest pusty String. Jeśli parsowanie i wstawianie danych przebiegło bez problemu, do gniazda linia po linii pisane są id nowych ścieżek. Następnie pisane są informacje o markach (punktach specjalnych), jeśli takie były w pliku. Każda linia zawiera opis jednego punktu i ma postać: "#szerokość_geograficzna:długość_geograficzna:typ:ewentualny_opis_słowny".

Aby usunąć ścieżkę o podanym numerze do gniazda należy wpisać w pierwszej lini "delete", a w kolejnej identyfikator trasy.

5.4 Przetwarzanie danych

5.4.1 Upraszczanie ścieżek

Podczas rejestracji trasy za pomocą modułu *Gilgamesh* powstaje plik, zawierający współrzędne geograficzne punktów. Częstotliwość zapisu ustala użytkownik. Niezależnie od jego wyboru, w pliku zawsze znajdują się punkty nadmiarowe, niewnoszące wkładu do kształtu ścieżki. Podczas wstawiania do bazy danych punkty zostają więc przefilrowane, aby nie przechowywać niepotrzebnych.

Odpowiedzialna jest za to klasa *PolylineSimplifier*. Na początku przeprowadzana jest iteracja po wszytkich punktach. W każdym kroku obliczana jest odległość punktu od następnego. Jeśli jest zbyt mała, to ten następny punkt zostaje usunięty. Ta metoda eliminuje ze ścieżki skupiska powstałe np. w wyniku postoju na trasie, kiedy odbiornik GPS poruszał się jedynie nieznacznie i rejestrował właściwie jedno miejsce.

Następnie ścieżka może zostać podzielona na kilka krótszych. Podczas rejestracji trasy GPS może stracić sygnał satelitarny (problem ten wynika z ograniczeń sprzętowych odbiornika, ewentualnie systemu GPS, natomiast nie jest spowodowany błędami aplikacji mobilnej). Kiedy rejestracja trasy zostanie wznowiona, w pliku powstaje spora przerwa w odczytach, niezgodna z faktycznym przebiegiem drogi. Jeśli odległość między dwoma kolejnymi punktami w pliku jest kilkakrotnie większa niż lokalna średnia, to w tym miejscu ścieżka jest dzielona na dwie krótsze, które zostaną dalej uproszczone osobno.

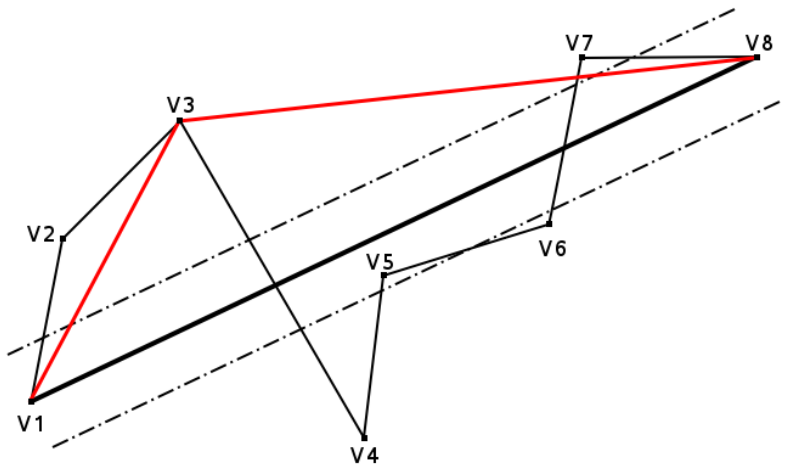
Kiedy odfiltrowane zostały niepotrzebne skupiska punktów i ścieżki są podzielone, dla każdej z nich wywołana zostaje najważniejsza metoda, redukująca wierzchołki pojedynczej łamanej. W tym celu zastosowaliśmy algorytm

Douglasa i Peuckera. Opiera się na podejściu całościowym, a nie lokalnej analizie trasy. Jest uznawany za jedną z solidniejszych kartograficznie metod generalizacji odcinków.

Do algorytmu potrzebny jest stos, na który odkłada się fragmanty łamanej do rozpatrzenia. Początkowo brana jest pod uwagę cała trasa. Wyznaczona zostaje linia łącząca punkt początkowy i końcowy oraz pas tolerancji wokół niej. Następnie obliczane są odległości wszystkich punktów pośrednich od tej prostej. Jeśli mieszczą się wewnątrz wyznaczonego pasa, zostają uznane za nieistotne i należy je usunąc z wynikowej ścieżki. Jeżeli natomiast odległości niektórych punktów nie mieszczą się w zakresie tolerancji, należy wybrać spośród nich najbardziej odległy. Wówczas na stosie zostają umieszczone dwie krótsze łamane, których końcami są najdalszy punkt i odpowiednio końce aktualnie rozpatrywanej łamanej. Algorytm rozpoczyna kolejną iterację, zdejmując ze stosu łamaną do rozpatrzenia.

Algorytm działa dopóki na stosie znajdują się jakieś odcinki. Wprowadziliśmy do niego jedynie drobną modyfikację: jeśli punkt jest markowany, to nie zostaje usunięty, nawet jeżeli wynikałoby to z kształtu ścieżki.

Zasadę działania algorytmu wyjaśnia rysunek. Widać, że wierzchołek V3 jest najbardziej oddalony od odcinka łączącego V1 i V8. Na stos trafiają więc łamane od V1 do V3 i od V3 do V8.



Rysunek 5.1: Algorytm Douglasa-Peuckera

5.4.2 Uśrednianie tras

Dowolne trasy mogą być wielokrotnie rejestrowane i wprowadzane do serwisu. Moduł *Hammurabi* podczas wprowadzania nowej ścieżki wykrywa, z jakimi przebywającymi w bazie może się pokrywać. Jeśli takie trasy się znajdą, dla każdych nakładających się (z zadaną dokładnością) odczytów obliczana jest średnia i obie trasy zostają odpowiednio zmodyfkowane. Dzięki temu, jeśli jakiś odcinek zostanie zarejestrowany kilkukrotnie, na mapie jest wyświetlany jako jedna łamana, a nie kilka przebiegów, które by się wzajemnie przecinały i zaciemniały obraz.

Dokładniej, po uproszczeniu ścieżki wprowadzanej do serwisu, znajdowane są trasy, które przebiegają przez teren o tych samych współrzędnych geograficznych. Nowy przebieg jest więc kolejno prównywany z każdym z potencjalych sąsiadów. Dla takich dwóch tras tworzona jest tablica odległości punktów, aby można było znaleźć najbliższe sobie odczyty w obu łamanych. Jeśli jakieś odcinki łączące są krótsze niż zadana z góry dokładność (wyznaczona doświadczalnie), to punkty są scalane, aby na mapie wyświetlane były w dokładnie tym samym punkcie. Jednocześnie, jeśli w ścieżkach znajdują się podobne punkty i zostają scalone, ustawiana jest odpowiednia flaga, informująca o konieczności modyfikacji ścieżki będącej już w bazie danych.

Metody scalania podobnych ścieżek zostały zaimplementowane w klasie *PolylineMerger*.

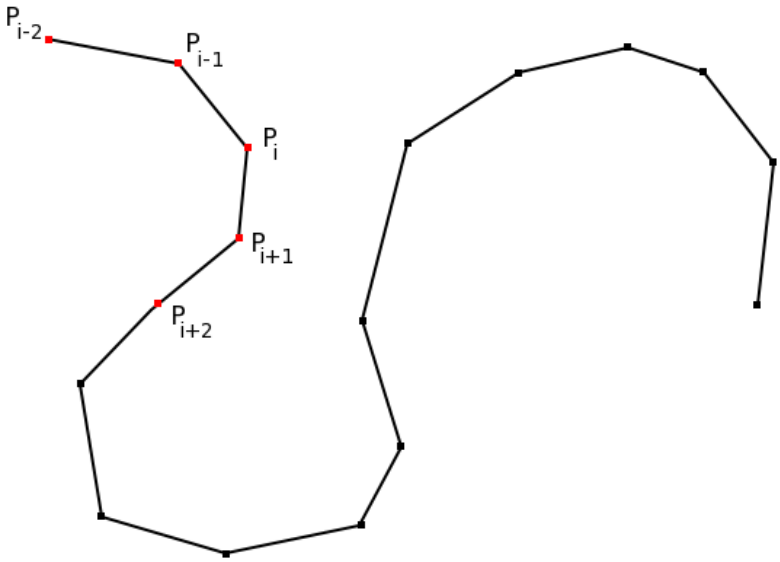
5.4.3 Wygładzanie ścieżek

Wygładzanie nie ma wpływu na dane przebywające w bazie. Używane jest podczas wydobywania ścieżek zarówno do modułu *Semiramida*, jak i *Marduk*. Polega na nieznacznym przesunięciu punktów ścieżki tak, aby wyglądała bardziej naturalnie. Nie prowadzi to do przekłamania danych, natomiast pozwala na lepsze przedstawienie wizualne. Opowiada za to klasa *PolylineSmoother*.

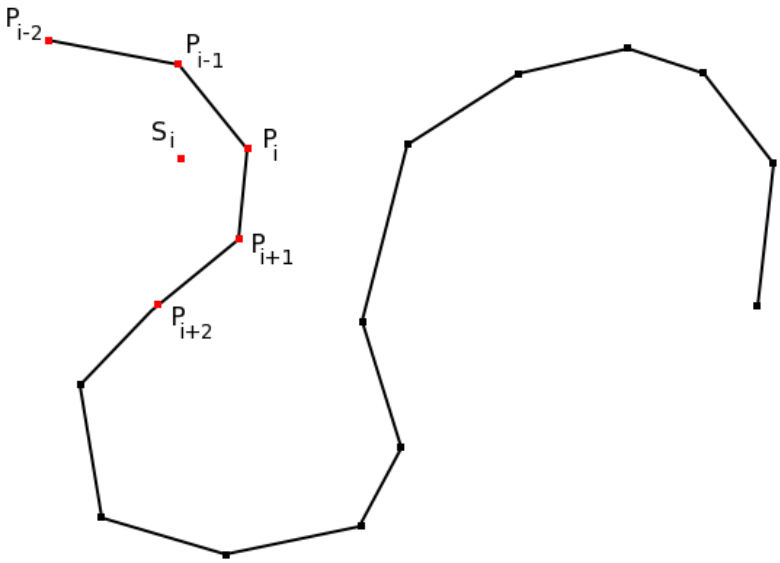
W celu wygładzania ścieżek zaimplementowany został algorytm ważonych odległości McMastera. Należy z góry określić, ile punktów ma wpływ na przesunięcie danego. Ustaliliśmy tę liczbę na 5. Dla punktów brzegowych brakuje sąsiadów, więc po dwa na końcach nie ulegają modyfikacji. Pozostałe punkty zostają wygładzone w iteracji: dla punktu P_i obliczona zostaje średnia ważona sąsiadów: od P_{i-2} do P_{i+2} (zgodnie z zasadą, że im dalszy punkt, tym mniejszy ma wpływ na średnią):

$$S_i = \frac{\frac{1}{3}P_{i-2} + \frac{1}{2}P_{i-1} + P_i + \frac{1}{2}P_{i+1} + \frac{1}{3}P_{i+2}}{\frac{1}{3} + \frac{1}{2} + 1 + \frac{1}{2} + \frac{1}{3}} \tag{5.1}$$

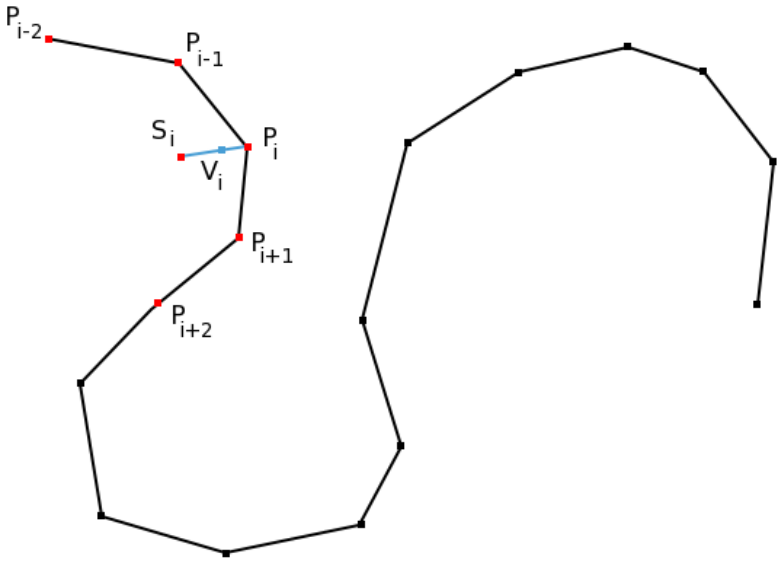
Mając wyznaczony w ten sposób punkt S_i, wynikowy obliczany jest ponownie jako średnia ważona pomiędzy S_i a P_i (siła przyciągania obu punktów ustalona została empirycznie).



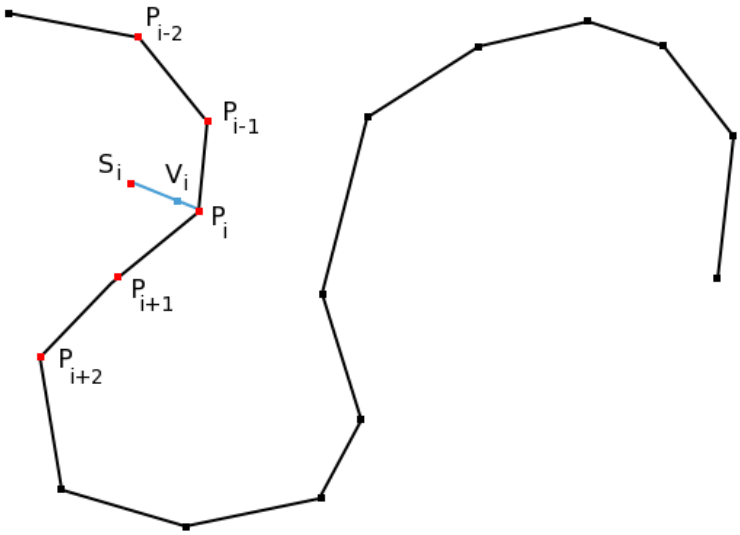
(a) Do wygładzenia P_i brane są pod uwagę również punkty sąsiednie



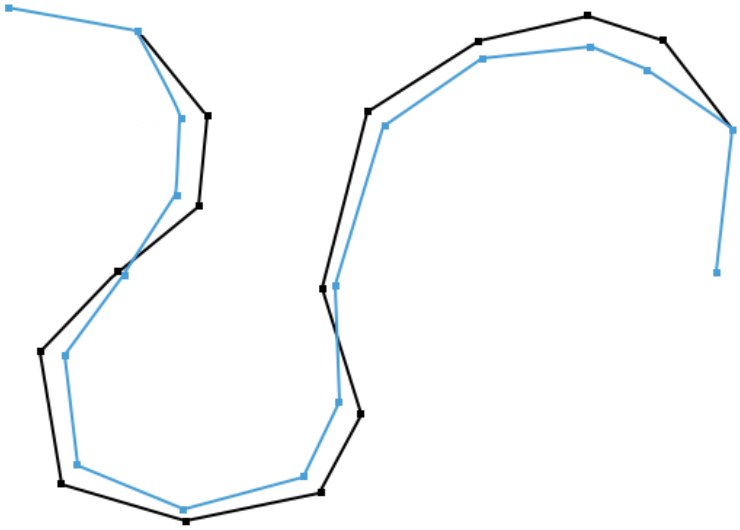
(b) Obliczona zostaje średnia ważona punktów



(c) Punkty P_i i S_i łączone są odcinkiem, wynikowy V_i jest ich średnią, zależną od siły przyciągania P_i



(d) Przesunięcie kolejnego punktu



(e) Ścieżka po wygładzeniu

Rysunek 5.2: Algorytm McMastera

Rozdział 6

Semiramida

Centralną częścią projektu Nabuchodonozor jest moduł Semiramida, czyli portal internetowy, którego zadaniem jest wizualizacja tras użytkowników, przy wsparciu Google Maps.

Największą zaletą tego portalu jest powszechny dostęp do tras nieoznaczonych na mapach, jakości ich nawierzchni oraz lokalizacji miejsc wartych zwiedzenia. Ponadto moduł ten umożliwia filtrowanie wyświetlanych tras, wyszczególnianie ich określonych cech, dodawanie komentarzy i zdjęć oraz eksportowanie do różnych formatów.

6.1 Decyzje projektowe

6.1.1 Google Maps

Semiramida do wizualizacji danych korzysta z *Google Maps API*. Wybór tego środowiska podyktowany jest następującymi powodami:

- *Google Maps* zawierają dane kartograficzne niemal dla całego globu. Znacząca część tych danych jest wysokiej jakości, umożliwia to łatwą identyfikację obiektów. Poza zdjęciami satelitarnymi zawiera również klasyczne mapy z nazwami miejscowości, ulic, rzek.

- Do *Google Maps* dołączone jest wygodne i wielofunkcyjne *API*. Umożliwia ono wyświetlanie map na własnych stronach internetowych. Jedną z największych zalet tego *API* jest możliwość stosowania *nakładek* (ang. *overlay*). Umożliwiają one wyświetlanie ścieżek i znaczników.

- Inną ważną zaletą *Google Maps* jest ich popularność. To najbardziej znany serwis tego typu, więc nie powinien sprawiać problemów w obsłudze.

- Warto również wspomnieć o *Google Earth*. Ten darmowy program rozszerza możliwości *Google Maps* o wizualizację trójwymiarową. Pozwala również na większą kontrolę nad parametrami wyświetlania danych. Odbywa się to kosztem wiekszych wymagań systemowych oraz konieczności pobrania i zainstalowania programu.

  Należy również wspomnieć o tym, że *Nabuchodonozor* umożliwia eksport do formatu *KML*, który jest natywnym formatem *Google Earth*.

6.1.2 Python

Python to wysokopoziomowy język programowania. Jest on wyjątkowo zwarty. Wśród jego zalet należy wymienić automatyczne zarządzanie pamięcią, dostępność wielu biblotek oraz jednoczesne spełnianie kilku paradygmatów.

Język ten ma silne wsparcie dla programowania obiektowego. Poosiada w pełni dynamiczny system typów (stosuje *duck typing* — „jeśli chodzi i mówi jak kaczka, oznacza to, że jest kaczką", co eliminuje konieczność stosowania interfejsów).

6.1.3 Django

Django to zdobywajacy coraz większą popularność framework służący do tworzenia aplikacji internetowych.

Jest odpowiedzią środowiska programistów Pythona na framework Ruby on Rails. Mimo, że jego pierwowzorowi nie brakuje wiele do doskonałości, to samo Django jest znaczącym krokiem w przód.

Oto najważniejsze cechy *Django*, z których korzystamy:

- przyjazne adresy dokumentów
- stosowanie wzorca projektowego MVC
- prosty, przyjazny użytkownikowi i jednocześnie funkcjonalny system szablonów
- wszechstronne wsparcie dla wielojęzycznych aplikacji
- wyjątkowo przyjazny, ale posiada również silne mapowanie obiektowo-relacyjne (ORM - object-relational mapping)
- wsparcie dla popularnych baz danych, włączając w to *PostgreSQL* i *MySQL*

Uwaga: Pomimo tego, że *Django* korzysta z wzorca MVC (Model-View-Controller) czyli Model-Widok-Sterownik to stosuje względem niego odmienne nazewnictwo. Mowa tu o MTV (Model-Template-View) czyli Model-Szablon-Widok. Takie oznaczenie może początkowo być mylące, zwłaszcza, że to co w MVC jest nazywane widokiem w MTV nazywane jest szablonem.

Aby nie powodować niejednoznaczności przyjmę oznaczenia stosowane przez twórców frameworka.

6.2 Instalacja

Instalacja *Semiramidy* zostanie omówiona na przykładzie systemu *GNU/Linux* oraz bazy danych *PostgreSQL*. Instalacja w systemie MS Windows przebiega w tej samej kolejności, różni się jednak wykorzystaniem gotowych pakietów binarnych.

Należy zwrócić uwagę, że instalujemy *Pythona 2.4* oraz *psycopg 1*. Wybór tej platformy jest wynikiem tego, że te wersje zostały gruntownie przetestowane pod względem współpracy z *Django*. Należy jednak wspomnieć, że obecnie możliwe jest również korzystanie z *Pythona 2.5* (który z kolei wymaga *psycopg 2*).

6.2.1 Baza danych

Przebieg instalacji bazy danych jest przedstawiony w dokumentacji technicznej modułu *Hammurabi*. Jako że obie aplikacje korzystają z tej instancji bazy danych więc można założyć, że baza jest już zainstalowana.

6.2.2 Python

Instalacja *Pythona* przebiega standardowo, więc zostanie przedstawiona w skrócie.

```
wget http://www.python.org/ftp/python/2.4.4/Python-2.4.4.tgz
tar -zxvf Python-2.4.4.tgz
cd Python-2.4.4
./configure
make
make install
```

6.2.3 Python Imaging Library

Biblioteka *Python Imaging Library*, czyli w skrócie *PIL*, jest wykorzystywana w *Semiramidzie* do zmiany rozmiaru zdjęć ilustrujących znaczniki.

Instalacja sprowadza się do wydania następujących poleceń:

```
wget http://effbot.org/downloads/Imaging-1.1.6.tar.gz
tar -zxvf Imaging-1.1.6.tar.gz
cd Imaging-1.1.6
python setup.py install
```

6.2.4 psycopg

Biblioteka *psycopg* umożliwia komunikację pomiędzy *Pythonem* a *PostgreSQL*. Jednak aby móc ją zainstalować, musimy mieć w systemie *eGenix.com mx Extensions for Python*.

Jeśli jej brak to należy ją zainstalować nastepującymi poleceniami:

```
wget http://www.egenix.com/files/python/egenix-mx-base-2.0.6.tar.gz
tar -zxvf egenix-mx-base-2.0.6.tar.gz
cd egenix-mx-base-2.0.6
python setup.py install
```

Teraz możemy już przejść do właściwej instalacji:

```
wget http://www.initd.org/pub/software/psycopg/psycopg-1.1.21.tar.gz
tar -zxvf psycopg-1.1.21.tar.gz
cd psycopg-1.1.21
./configure
make
make install
```

6.2.5 Django

Gdy podane etapy mamy wykonane, możemy przejść do instalacji *Django*. Ponieważ ten framework rozwija się bardzo szybko, pobierzemy wersję rozwojową. Musimy w tym celu skorzystać z *svn*, który musi znajdować się w systemie.

Aby zainstalować *Django* wydajemy nastepujące polecenia:

```
svn co http://code.djangoproject.com/svn/django/trunk/ django_src
ln -s `pwd`/django_src/django /usr/lib/python2.4/site-packages/django
```

Uwaga Aby zaktualizować *Django* do najnowszej wersji wystarczy przejść do katalogu `django_src` i wykonać polecenie:

```
svn update
```

6.2.6 Sprawdzenie instalacji

Aby zweryfikować poprawność instalacji należy wydać nastepujące polecenia:

```
% python
>>> import psycopg
>>> import PIL
>>> import django
```

Jeśli wykonają się bezbłędnie oznacza to, że instalacja przebiegła pomyślnie i można przejść do instalacji *Semiramidy*

6.2.7 Semiramida

Po pobraniu Semiramidy ze strony WWW należy ją rozpakować ją do katalogu, w którym ma pracować. Później łatwo zmienić miejsce instalacji, lecz wymaga to ponownej edycji pliku konfiguracyjnego.

Następnie należy dokonać edycji pliku konfiguracyjnego. Konfiguracja jest dokładniej opisana w rozdziale **Konfiguracja**. Z najważniejszych opcji należy zmodyfikować `SEMIRAMIS_HOST_PREFIX`, `SEMIRAMIS_BASE_DIR` `SEMIRAMIS_GOOGLE_KEY` oraz wszystkie rozpoczynające się od `DATABASE_`.

Po upewnieniu się, że baza danych działa, należy stworzyć strukturę bazy danych poleceniem `./manage.py syncdb`. Polecenie to ponadto zasugeruje stworzenie administratora serwisu, z czego należy skorzystać.

W skrócie proces instalacji można przedstawić następująco:

```
wget http://any.server.pl/path/to/nebuchadrezzar.tar.gz
tar -zxvf nebuchadrezzar.tar.gz
cd nebuchadrezzar
vim settings.py
./manage.py syncdb
```

Teraz można już uruchomić *Semiramidę* na uprzednio wybranym adresie i porcie. Robi się to następującym poleceniem:

```
./manage.py runserver
```

Uwaga 1 Do uruchomienia *Semiramidy* nie jest konieczny pracujący *Hammurabi*. Można go uruchomić po uruchomieniu serwisu internetowego. Brak pracującego *Hammurabiego* powoduje niemożliwość oglądania ścieżek na mapie, oraz brak dostępności eksportów danych.

Uwaga 2 W zastosowaniach produkcyjnych zalecane jest korzystanie z serwera *Apache* z *mod_python*. Konfiguracja taka jest opisana na stronie *Django*.

6.3 Konfiguracja

6.3.1 Plik konfiguracyjny

Pierwszą rzeczą, jaką trzeba wykonać po instalacji *Semiramidy*, jest jej konfiguracja. Cała konfiguracja znajduje sie w jednym pliku `nebuchadrezzar/settings.py`. Tak jak sugeruje nazwa, jest to standardowy plik źródłowy *Pythona*. Ma to szereg zalet, takich jak czytelność czy możliwość stosowania zaawansowanych wyrażeń do ustalania opcji.

Najważniejsze wpisy, które trzeba można to:

```
DEBUG = True
TEMPLATE_DEBUG = DEBUG

SEMIRAMIS_HAMMURABI_HOST = ''
SEMIRAMIS_HAMMURABI_PORT = 1313

SEMIRAMIS_ITEMS_PER_PAGE = 10
SEMIRAMIS_ADJACENT_PAGES = 5
SEMIRAMIS_PHOTO_THUMB = (100,100)
SEMIRAMIS_PHOTO_LARGE = (480,480)
SEMIRAMIS_DATE_FORMAT = '%m-%d-%Y %H:%M:%S'

SEMIRAMIS_HOST_PREFIX = 'http://10.0.0.7:8080'
SEMIRAMIS_DIR_PREFIX = ''

SEMIRAMIS_BASE_DIR = '/opt/nebuchadrezzar/'
SEMIRAMIS_PHOTO_DIR = join(SEMIRAMIS_BASE_DIR, 'media/photo')
SEMIRAMIS_MEDIA_DIR = join(SEMIRAMIS_BASE_DIR, 'media')
SEMIRAMIS_STATIC_DIR = join(SEMIRAMIS_BASE_DIR, 'static')

SEMIRAMIS_GOOGLE_KEY = {
```

```
  "localhost:8080":  "ABQI...LatLQ",
  "10.0.0.7:8080":   "ABQI...z91Tg",
  "example.com:8000":"ABQI...GAmag",
}

TEMPLATE_DIRS = (
  join(SEMIRAMIS_BASE_DIR, 'template'),
)

DATABASE_ENGINE   = 'postgresql'
DATABASE_NAME     = 'dbname'
DATABASE_USER     = 'dbuser'
DATABASE_PASSWORD = 'dbpassword'
DATABASE_HOST     = ''
DATABASE_PORT     = ''
```

Jak łatwo zauważyć, część opcji zaczyna się od `SEMIRAMIS_`. Tak oznaczone są opcje specyficzne dla *Semiramidy*. Pozostałe parametry pozbawione tego prefiksu są standardowymi ustawieniami *Django*.

Znaczenie poszczególnych opcji:

`DEBUG` Włącza tryb debugowania aplikacji. Gdy jest ustawione, *Django* nie pomija po cichu błędów, lecz informuje o miejscu i prawdopodobnym powodzie ich wystąpienia. Warto zwrócić uwagę, że szablony błędów *HTTP 404* oraz *HTTP 500* są wyświetlane tylko wtedy gdy ta opcja jest ustawiona na `False`.

`TEMPLATE_DEBUG` Włącza tryb debugowania szablonów. Działa analogicznie jak `DEBUG`, lecz w odniesieniu do szablonów.

`SEMIRAMIS_HAMMURABI_HOST` Adres maszyny, na której działa *Hammurabi*. Można stosować zarówno adresy IP (przykładowo: `'192.168.1.5'`) jak i nazwy hostów (przykładowo: `'dbserver'`). Pusty napis (`''`) oznacza lokalną maszynę(*localhost*).

`SEMIRAMIS_HAMMURABI_PORT` Port TCP na którym działa *Hammurabi*. Domyślnie jest to `1313`.

`SEMIRAMIS_ITEMS_PER_PAGE` Ilość ścieżek lub znaczników, jaka ma się pojawić jednocześnie na stronie. Wartość ta jest odczytywana przez klasę `Pager` jeśli w jej konstruktorze nie zostanie podane inaczej.

`SEMIRAMIS_ADJACENT_PAGES` Na stronach wyświetlających listę ścieżek lub znaczników znajdują się przyciski umożliwiające natychmiastowe przejście do sąsiednich stron. Ta opcja określa, ile maksymalnie może być ich wyświetlonych. Oczywiście jeśli stron jest mniej niż podana liczba, to odnośniki do nich nie zostaną wyświetlone.

`SEMIRAMIS_PHOTO_THUMB` Maksymalny rozmiar, jaki może przyjąć miniaturka zdjęcia. Jest to również rozmiar, jaki przyjmie przycięte zdjęcie. Wartość ta jest podana w postaci krotki, której pierwszym elementem jest szerokość a drugim długość.

`SEMIRAMIS_PHOTO_LARGE` Maksymalny rozmiar, jaki może przyjąć standardowa wersja zdjęcia. Wartość jest interpretowana analogicznie jak w `SEMIRAMIS_PHOTO_THUMB`.

`SEMIRAMIS_DATE_FORMAT` Napis formatujący używany przez funkcję `strftime` biblioteki standardowej Pythona. Jest używany przez filtr `date_format` w systemie szablonów *Semiramidy*. Dzięki temu w łatwy sposób można zmienić wyświetlany format czasu i daty, dokonując edycji tylko jednego pliku.

`SEMIRAMIS_HOST_PREFIX` Adres maszyny, na której działa *Semiramida*. To ustawienie jest wykorzystywane przez funkcję `global_link`. Ta funkcja jest używana zgodnie ze swoją nazwą, czyli do tworzenia pełnych odnośników.

`SEMIRAMIS_DIR_PREFIX` Podkatalog, URL w którym ma być widoczna Semiramida.

`SEMIRAMIS_BASE_DIR` Absolutna ścieżka do katalogu `nebuchadrezzar`, czyli miejsca zainstalowania aplikacji.

`SEMIRAMIS_PHOTO_DIR` Absolutna ścieżka do katalogu `nebuchadrezzar/media/photo`, czyli miejsca gdzie są przechowywane zdjęcia przypisane do znaczników.

`SEMIRAMIS_MEDIA_DIR` Absolutna ścieżka do katalogu `nebuchadrezzar/media`, czyli miejsca przechowywania plików multimedialnych.

`SEMIRAMIS_STATIC_DIR` Absolutna ścieżka do katalogu `nebuchadrezzar/static`, czyli miejsca gdzie znajdują się statyczne pliki dostępne z poziomu przeglądarki. Mowa tu o plikach stylów (*CSS*), JavaScript i elementach graficznych strony.

`SEMIRAMIS_GOOGLE_KEY` Słownik zawierający nazwy hostów i powiązane z nimi klucze do *Google Maps API*.

`TEMPLATE_DIRS` Krotka zawierająca absolutne ścieżki do katalogów zawierających pliki szablonów *Django*.

`DATABASE_ENGINE` Silnik bazodanowy, z jakiego ma korzystać *Semiramida*. Jedna z poniższych wartości:

`'postgresql'` baza danych *PostgreSQL* korzystająca z *psycopg*

`'postgresql_psycopg2'` baza danych *PostgreSQL* korzystająca z *psycopg2*

`'mysql'` baza danych *MySQL*

`DATABASE_NAME` Nazwa bazy danych. Baza danych musi uprzednio istnieć (*Django* jej nie stworzy).

`DATABASE_USER` Nazwa użytkownika bazy danych, na którego koncie ma działać *Semiramida*.

`DATABASE_PASSWORD` Hasło dostępu do bazy danych dla wybranego użytkownika.

`DATABASE_HOST` Adres maszyny, na której działa baza danych. Można stosować zarówno adresy IP (przykładowo: `'192.168.1.5'`) jak i nazwy hostów (przykładowo: `'dbserver'`). Pusty napis (`''`) oznacza lokalną maszynę(*localhost*).

`DATABASE_PORT` Numer portu, na którym działa baza danych. Pusty napis (`''`) oznacza domyślny port dla wybranego silnika bazodanowego.

6.3.2 Klucz Google Maps API

Aby móc korzystać z mapy, należy udać się na stronę `google.com/apis/maps/signup.html` i podać adres, pod jakim ma być widoczna strona. Wygenerowany klucz należy umieścić w pliku konfiguracyjnym w słowniku `SEMIRAMIS_GOOGLE_KEY` według dostępnego tam wzoru.

Należy zwrócić uwagę, że klucz jest sprawdzany pod względem zgodności z adresem strony, którą użytkownik poda w przeglądarce. Oznacza to, że z punktu widzenia *Google Maps API* adresy `http://1.2.3.4/map/` oraz `http://example.com/map/` to zupełnie inne adresy, nawet gdy prowadzą do tego samego dokumentu. Oznacza to, że mapa dysponująca kluczem tylko dla jednego adresu nie będzie działała, gdy sie przejdzie do niej korzystając

z drugiego. Na szczęście można wygenerować wiele kluczy, po jednym dla każdego adresu, pod którym strona ma być dostępna.

6.4 Model

Model to część aplikacji, która umożliwia korzystanie z bazy danych. W *Semiramidzie* opiera się on w całości na systemie dostarczanym przez *Django*. Dzięki temu jest wyjątkowo prosty w użyciu jak i łatwy do zrozumienia.

Model deklaruje się tworząc klasy dziedziczące z klasy `django.db.models.Model`. Pojedyncza klasa odpowiada tabeli. Za pomocą pól w klasie deklaruje się kolumny w bazie.

Przykładowa klasa deklarująca tabelę `Track`:

```
class Track(models.Model):
  key = models.IntegerField(unique=True, blank=False, null=False)
  user = models.ForeignKey(User)
  name = models.CharField(blank=False, maxlength=100)
  text = models.TextField(blank=True)
  date = models.DateTimeField(auto_now_add = True)
  color = models.CharField(blank=True, default='', maxlength=3)
  def __str__(self):
    return self.name
  class Meta:
    ordering = ('-date',)
```

6.4.1 Opis tabel

Ponieważ tabele w *ORM Django* są opisane za pomocą klas, których nazwy i parametry są samoopisujące, to tabele zostaną tylko wymienione.

`Track` Opis ścieżek.

`TrackComment` Komentarze do ścieżek.

`Mark` Opis znaczników.

`MarkComment` Komentarze do znaczników.

`MarkPhoto` Zdjęcia do znaczników.

W celu uzyskania większej ilości informacji należy przejrzeć plik `nebuchadrezzar/semiramis/models.py`.

6.4.2 Pozostale tabele

Poza wymienionymi powyżej tabelami *Semiramida* korzysta również z innych. Mowa między innymi o tabelach odpowiadających za obsługę użytkowników. Nie zostaną one tu omówione, gdyż nie są bezpośrednio wykorzystywane. Ich omówienia należy szukać na stronach projektu *Django*.

6.5 Szablony

Semiramida korzysta z systemu szablonów dołączonego do *Django*. Dzięki temu udało sie osiągnąć znaczna separację kodu *HTML* od *Pythona*.

Szablony znajdują się w katalogu `nebuchadrezzar/template`. Są to zasadniczo zwykłe pliki *HTML*, które można edytować w zwykłych edytorach tekstu. Przykładowy, uproszczony szablon:

```
{% extends "base.html" %}
{% load i18n %}
{% load semiramis_templatetags %}

{% block title %}{% trans "Browse" %}{% endblock %}

{% block content %}
  {% for item in items %}
    <a href='{{ item|link_to }}'>
      <h1>{{ item.name|escape }}</h1>
    </a>
    {{ item.text|babel:"head" }}
  {% endfor %}
{% endblock content %}
```

Jedną z intensywniej wykorzystywanych cech jest *dziedziczenie szablonów*. Dzięki temu łatwo jest stosować zasadę *DRY* (*Don't Repeat Yourself*). Struktura dokumentu zdeklarowana jest w pliku `nebuchadrezzar/template/base.html`. Zdecydowana większość pozostałych szablonów tylko wypełnia niektóre pola w bazowym.

6.6 Widok

Widok jest elementem spajającym pozostałe dwa elementy MTV. Zawiera również całą logikę aplikacji. Zdecydowana większość funkcji widoku znajduje

się w pliku `nebuchadrezzar/semiramis/views.py`, a funkcje pomocnicze można znaleźć w `nebuchadrezzar/semiramis/utils.py`.

6.6.1 Kanały RSS i Atom

Semiramida umożliwia powiadamianie o nowo dodanych danych za pomocą kanałów. Korzysta w tym celu z *Django*, dzięki czemu instalacja bezproblemowo udostępnia je nie tylko w obecnie najpopularniejszym formacie *RSS*, ale również *Atom*.

Klasy obsługujące kanały znajdują się w pliku `nebuchadrezzar/semiramis/feeds.py`.

6.6.2 XML-RPC

Dostęp do danych nie jest ograniczony do mapy na stronie czy możliwości eksportu danych do popularnych formatów. Dla programistów *Semiramida* udostępnia bardzo wygodną i prostą alternatywę.

XML-RPC, bo o nim tu mowa, to protokół *RPC* (*Remote Procedure Call — protokół zdalnego wywoływania procedur*) oparty o język *XML*. Korzystanie z niego jest wyjątkowo proste, za przykład warto podać poniższy program:

```
import xmlrpclib
rpc = xmlrpclib.ServerProxy("http://example.com:8080/xmlrpc/")
for i in rpc.getTracks(0, 10):
  print i['id'], '>>', i['name']
```

Program ten pobiera z *Semiramidy* (działającej na adresie `example.com:8080`) 10 pierwszych ścieżek, wraz z identyfikatorem i nazwą, a nastepnie wyświetla te dane.

W chwili obecnej dostępne są następujące funkcje:

- getTracks
- getMarks
- getTrackPoints

Z łatwością można dodać następne. W tym celu należy dokonać edycji pliku `nebuchadrezzar/semiramis/xmlrpc.py` tworząc wymaganą funkcję, a następnie rejestrując ją funkcją:

```
dispatcher.register_function(newFunction, 'newFunction')
```

6.7 Mapowanie URL

Jedną z najszybciej dostrzegalnych zalet wynikających ze stosowania *Django* jest to, że stosuje on przyjazne adresy *URL*. Oznacza to, że używane są adresy udające ścieżki katalogów (typu `/mark/123/`) a nie parametry zapytania GET (których odpowiednik dla poprzedniego przykładu mógł by wyglądać tak:`/index.php?view=mark&id=123`).

Ma to szereg zalet, z których warto wymienić:

- lepiej, klarowniej wyglądające adresy
- łatwiejsze do zapamiętania i zapisania adresy
- lepsza dostępność dla robotów internetowych indeksujących strony internetowe

6.7.1 Mapowanie adresu na widok

Zadaniem uruchomienia właściwej (przypisanej do danego adresu) funkcji widoku zajmuje się *Django*. Do tego wymaga jednak odpowiedniego pliku. W *Semiramidzie* jest to `nebuchadrezzar/semiramis/urls.py`. Przykładowe kilka linii z tego pliku:

```
from django.conf.urls.defaults import *

urlpatterns = patterns('nebuchadrezzar.semiramis',
  (r'^$',                     'views.main'),
  (r'^map/$',                 'views.gmap'),

  (r'^track/$',               'views.track_page'),
  (r'^track/page([0-9]+)/$', 'views.track_page'),
  (r'^track/([0-9]+)/$',      'views.track_id'),
)
```

Jak widać, jest to zupełnie standardowy plik *Pythona*. Jest on doskonale opisany w dokumentacji frameworka więc w razie wątpliwości należy z niej skorzystać.

6.7.2 Mapowanie obiektów i widoków na adres

Django nie dostarcza żadnej funkcji, podającej adres, pod którym dostępny jest dany widok lub obiekt. Wprawdzie autorzy frameworka sugerują dodawanie metod (w klasach modelu), zwracających adres do danego obiektu (mowa o metodzie `get_absolute_url()`) ale to rozwiązanie ma szereg wad:

- brak rozdziału pomiędzy **widokiem** a **modelem**
- brak możliwości decyzji o tym czy odnośnik ma być podany w postaci adresu absolutnego czy względem katalogu głównego serwera
- brak możliwości podania dodatkowych opcji do umożliwiających np. stworzenie odnośnika do strony edycji danego obiektu
- umożliwia tworzenie odnośników tylko do obiektów które mają swoją reprezentację w bazie danych.

Wszystkich tych wad pozbawiona jest funkcja `link_to` i jej małe rozszerzenie pod nazwą `global_link`. Obie funkcje dostępne są w systemie szablonów, obie pobierają identyczne argumenty, które są identycznie interpretowane.

Różnica pomiędzy tymi funkcjami to fakt, że `link_to` zwraca adres względem katalogu głównego serwera a `global_link` wywołuje funkcję `link_to` i dokleja jeszcze na początku wyniku adres maszyny (podany w `SEMIRAMIS_HOST_PREFIX`, w pliku ustawień).

6.8 Wersje językowe

Semiramida jest przystosowana do obsługi wielu wersji językowych. W tym celu wykorzystuje wbudowane w język *Python* możliwości. Ponadto *Django* rozszerza je o *leniwe wartościowanie* co, pomaga przykładowo w tworzeniu formularzy.

Istotne jest to, że w tym celu wykorzystywana jest standardowa i popularna biblioteka *gettext*. Dzięki temu tłumacze nie muszą się zapoznawać z nowym środowiskiem. Ponadto istnieje wiele programów wspierających ich pracę.

Warto też wspomnieć o możliwości korzystania z tłumaczeń w szablonach. Ta funkcjonalność jest powszechnie wykorzystywana w *Semiramidzie*. Dzięki temu zwiększa sie rozdział widoku od szablonów.

6.8.1 Edycja tekstów źródłowych

Zasadniczo teksty w *Semiramidzie* są w języku angielskim, co ułatwia tłumaczenie serwisu na inne języki. W razie chęci rozbudowy, przekształceń lub poprawy błędów w angielskim tekscie należy dokonać edycji plików szablonów (zawartość katalogu `nebuchadrezzar/template`) oraz plików `nebuchadrezzar/semiramis/views.py` i `nebuchadrezzar/semiramis/feeds.py`.

Wykorzystywane konstrukcje:

`gettext('text')` **oraz** `_('text')` Standardowe dla biblioteki *gettext* metody oznaczenia tekstu do przetłumaczenia

`gettext_lazy('text')` Udostępniana przez *Django* metoda, oznaczająca tekst do przetłumaczenia. Ma tę przewagę nad standardowymi, że korzysta z *leniwego wartościowania*. Dzięki temu może być wykorzystywana w deklaracji klas. W *Semiramidzie* jest to wykorzystywane do tłumaczenia nazw pól w formularzach.

`{% trans "text" %}` Podstawowa metoda tłumaczenia napisów w szablonach.

`{% blocktrans %}text{% endblocktrans %}` Rozszerzona metoda tłumaczenia napisów w szablonach. Pozwala na stosowanie pól wstawiających wartość zmiennych.

6.8.2 Wybór wersji językowej

Gdy już istnieją pliki wersji językowych, to pozostaje jeszcze jeden ważny etap. Mianowicie należy zdecydować, w jakim języku należy wysłać stronę konkretnemu użytkownikowi.

Semiramida korzysta z następującego algorytmu wyboru wersji językowej:

- W pierwszej kolejności jest używany język, którego kod znajduje się w *cookie* `django_language`.
- Jeśli poprzedni punkt nie przyniósł rozstrzygnięcia, to wykorzystywany jest *nagłówek HTTP* `Accept-Language`
- Ostatecznie wykorzystywana jest opcja `LANGUAGE_CODE` z pliku konfiguracyjnego (`settings.py`)

6.8.3 Tworzenie nowej wersji językowej

Aby stworzyć nową wersję językową (w tym przykładzie załóżmy, że niemiecką), należy wykonać w katalogu `nebuchadrezzar/` wykonać następujące polecenie:

```
./make-messages.py -l de
```

Teraz w katalogu `nebuchadrezzar/locale/en/LC_MESSAGES` pojawia się plik `django.po`. Jest to standardowy plik tłumaczeń, zgodny z systemem *gettext*. Do jego edycji można wykorzystać dowolny edytor tekstowy, posiadający obsługę *UTF-8* lub wykorzystać wyspecjalizowany edytor (przykładowo

KBabel). Gdy dokona się wymaganych tłumaczeń, należy skompilować pliki `po` do formatu `mo`:

```
./compile-messages.py
```

Jeśli odnajdzie się błędy lub inne niedoskonałości w wersji oryginalnej, to można je poprawić bez ponownego tłumaczenia całości (poprzednio przetłumaczone napisy są bezpieczne). Można to zrobić wywołując polecenie:

```
./make-messages.py -a
```

Po tym wystarczy dokonać edycji tłumaczeń i ponownie skompilować.

Rozdział 7

Marduk

Marduk jest modułem odpowiadającym za konwersję danych geograficznych pomiędzy różnymi formatami. Obejmuje import plików aplikacji *Nabuchodonozor* oraz parsowanie standardu NMEA czy GPX. Umożliwia również eksport do GPX, formatów drukowalnych (PDF i Postscript). Dodatkowym atutem jest funkcja konwersji do KML.

7.1 Format wewnętrzny aplikacji *Nabuchodonozor*

Dane zwracane przez moduł *Gilgamesh* mają postać plików tekstowych o specyficznym formacie. Rzeczą oczywistą jest, że pozostałe część aplikacji muszą obsługiwać ten format. Parsowaniem plików uzyskanych za pomocą modułu *Gilgamesh* z telefonów komórkowych zajmuje się klasa *TXTParser*. Parser czyta dane ze strumienia i wydobywa z nich współrzędne punktów oraz informacje o markach. Dane te zwracane są w postaci dwóch *ArrayList*.

Istnieje również możliwość eksportu danych do formatu tekstowego, aby umożliwić użytkownikom Gilgamesha korzystanie z gotowych paczek tras. Wówczas punkty ścieżek oraz marki zapisywane są w formacie omówionym wcześniej.

7.2 Import danych w formacie NMEA

NMEA jest standardem zapisu danych przez odbiorniki GPS. Aplikacja mobilna wchodząca w skład projektu Nabuchodonozor na bieżąco go parsuje, aby pobrać i zapisać tylko potrzebne informacje (działanie spowodowane koniecznością oszczędzania pamięci telefonu). W związku z tym użytkowni-

cy *Gilgamesha* wprowadzają do serwisu pliki tekstowe. Założyliśmy jednak, że nie każdy musi używać naszej aplikacji mobilnej. Istnieją bowiem droższe odbiorniki GPS, posiadające możliwość graficznej wizualizacji danych. Jeśli jednak takie urządzenie jest w stanie zapisywać przebieg trasy, nic nie stoi na przeszkodzie, aby te dane zaimportować do bazy danych.

Parser NMEA działa więc w dwóch miejscach - na urządzeniu mobilnym, parsując dane podczas pobierania z GPS, oraz w aplikacji bazodanowej, podczas wstawiania do bazy danych.

Parser NMEA 0183 oparty jest na zdarzeniach. Program chcący korzystać z *NmeaParser* musi posiadać klasę implementującą interfejs *GisInfo* - jest ona przekazywana parserowi podczas jego tworzenia. W zależności od napływających danych parser wywołuje odpowiednie metody z interfejsu.

Parser jest wzorowany na standardzie SAX. Realizuje abstrakcje źrodła danych. Nie zwraca informacji o odczytanym poleceniu NMEA, lecz o danych, jakie ona zawierała. Ułatwia to obsługę różnorodnych odbiorników, gdyż nie wszystkie muszą obsługiwać dane polecenia. Obsługuje najpopularniejsze, wystepujące niemal we wszystkich odbiornikach polecenia NMEA:

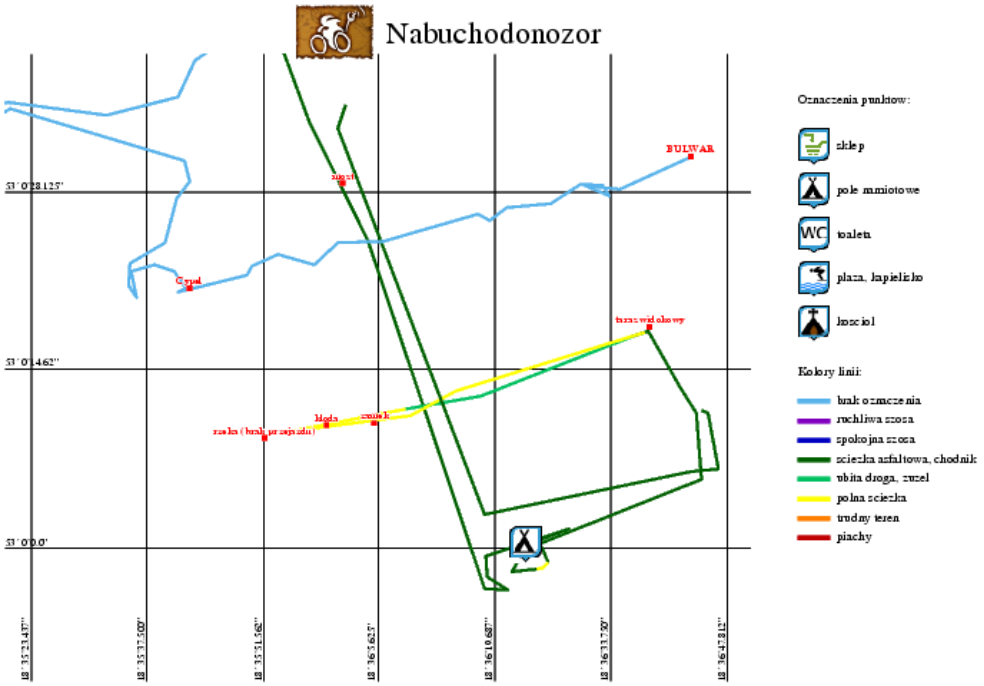
- GGA - fix information
- GLL - lat/lon data
- GSA - overall satellite data
- GSV - detailed satellite data
- RMC - recommended minimum data for gps

Jest odporny na uszkodzone dane, sprawdza sumy kontrolne, moze zostać podłączony do istniejącej już transmisji. Jeśli natrafi na jakiekolwiek błędy, po cichu je omija, nie próbujac odzyskiwać.

7.3 Eksport do Postscriptu

Postscript jest językiem programowania, opracawanym przez firmę Adobe Systems. Służy do opisu strony. Pozwala na dokładne określenie jej wyglądu, rozplanowanie elementów graficznych, a nawet wykonywanie operacji na otrzymanych danych. Pamięć oparta jest o definicję stosu, wszystkie operacje zapisywane są w notacji postfixowej. Obecnie Postscript uznawany jest za standard w zastosowaniach poligraficznych.

Przewidzieliśmy eksport ścieżek z bazy danych do formatu Postscript przede wszystkim ze względu na możliwość drukowania dokumentów i potencjalnego ich użycia podczas wypraw rowerowych. Dokument zawiera ścieżki



Rysunek 7.1: Eksport mapy do formatu PS

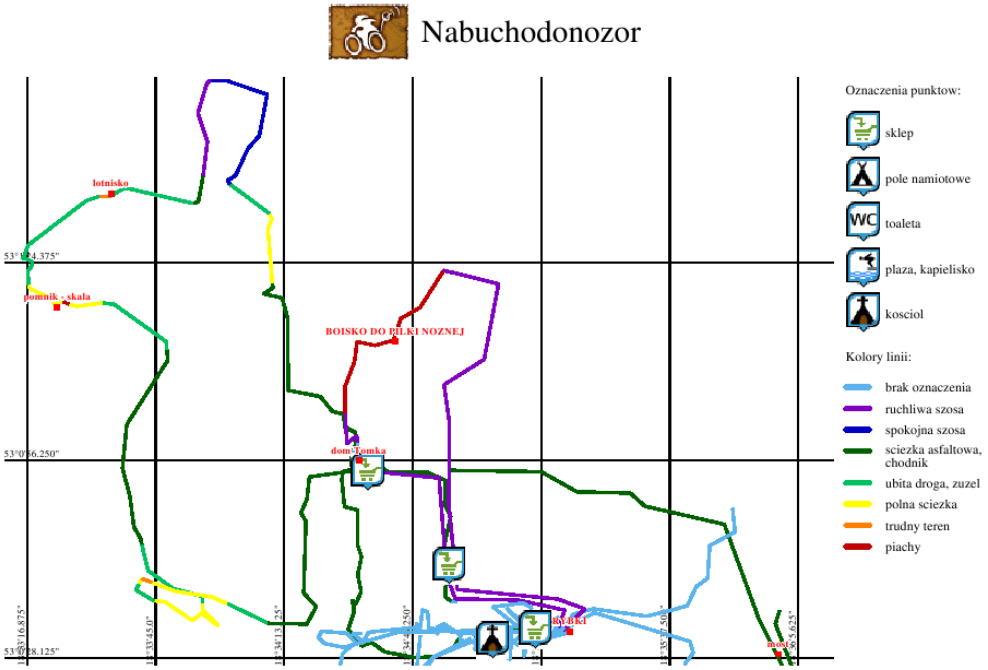
z danego obszaru globu, naniesione na siatkę południków i równoleżników. Punkty markowane zaznaczane są w postaci zaprojektowanych przez zespół ikon. Niestandardowe opatrzone są przypisami zrobionymi przez użytkowników *Gilgamesha*. Ponadto ścieżki mają kolory odzwierciedlające rodzaje nawierzchni. Mapa opatrzona jest legendą oraz logiem zespołu.

7.4 Eksport do PDF

Eksport do formatu PDF został zaimplementowany z tych samych powodów co Postscript, czyli przede wszystkim ze względu na możliwość wydruku. Dodatkowym czynnikiem był fakt, że Portable Document Format jest przenośny, na każdej platformie istnieją narzędzia do jego odczytu.

PDF ma budowę bardziej skomplikowaną niż Postscript, więc moduł *Marduk* nie konstruuje takiego dokumentu od podstaw. Dla ułatwienia pracy użyliśmy biblioteki iText dla języka Java. Jest to oprogramowanie autorstwa firmy Lowagie, rozprowadzane na licencji LGPL.

Użyta przez nas biblioteka posiada gotowe komponenty wspomagające tworzenie akapitów, tabel itp. Użyliśmy ich, nakładając na stronę nazwę projektu, logo zespołu oraz legendę. Właściwa mapa została skonstruowana przy użyciu komponenetów niskopoziomowych. Pozwoliło to na rysowanie ścieżek, południków i równoleżników z dokładnością do piksela. Podobnie, można w ten sposób dokładnie kontrolować miejsce wyświetlania ikon i opisów punktów. Dokument PDF wyeksportowany przez moduł *Marduk* jest bardzo podobny do opisywanego wyżej Postscriptu. Wybór formatu zależy jedynie od preferecji użytkownika.



Rysunek 7.2: Eksport mapy do formatu PDF

7.5 Formaty XML

Do obsługi formatów XML wykorzystaliśmy SAX (Simple API for XML). Jest to technologia oparta na zdarzeniach. Wykorzystana została do parsowania plików GPX i NMEA oraz tworzenia GPX i KML.

Do parsowania pliku potrzebny jst obiekt klasy *SAXParser*, który pobiera strumień zawierający dane w postaci XML. Jednocześnie wymagane jest

rozszerzenie klasy *DefaultHandler* i nadpisanie metod, w zależności od tego, jakie znaczniki i elementy planujemy obsłużyć:

- *startElement()* - metoda wywoływana, kiedy parser napotka znacznik rozpoczynający element w pliku XML; można pobrać jego nazwę i, w zależności od tego, odpowiednio obsłużyć; w GPX przechwytywane są informacje o rozpoczęciu ścieżki i punktu; jednocześnie dla punktu obiekt *Attributes* zawiera pola *lat* i *lng*, określające współrzędne geograficzne; punkt od razu wstawiany jest do aktualnie tworzonej ścieżki

- *characters()* - metoda jest wywoływana, kiedy parser napotka ciąg danych znakowych; jeśli akurat parsowany jest punkt, to dane znakowe mogą określać jego wysokość n. p. m.; wówczas w ścieżce punkt jest zmieniany, aby uzupełnić informacje o nim

- *endElement()* - metoda wykonuje się, jeżeli parser napotka znacznik końca elementu; jeśli kończona jest ścieżka, dodawana jest do listy wszystkich tras; pozostałe znaczniki są ignorowane.

Kiedy obiekt SAXParser napotka na znaczniki początku czy końca elementu, bądź ciąg danych znakowych, wywołuje jedną z powyższych metod. Dane z pliku są obsługiwane i utworzone zostają odpowiednie listy zawierające dane o ścieżkach i punktach markowanych.

Eksport danych do GPX i KML oparty jest o tę samą technologię. W tym wypadku najważniejszą rolę pełni obiekt klasy *TransformerHandler*, odpowiedzialny za tworzenie znaczników w XML-u. Osiągane jest to poprzez wywoływanie metod:

- *startDocument()* - stworzenie nowego dokumentu

- *startElement()* - nowy element w strukturze pliku XML

- *characters()* - wprowadza dane znakowe (np. wysokość punktu)

- *endElement()* - zakończenie elementu XML

- *endDocument()* - koniec dokumentu

Dodatkowo istotnym elementem jest klasa *AttributesImpl*, która odpowiada za atrybuty wprowadzanego do XML znacznika. Wykorzystywane jej dwie metody:

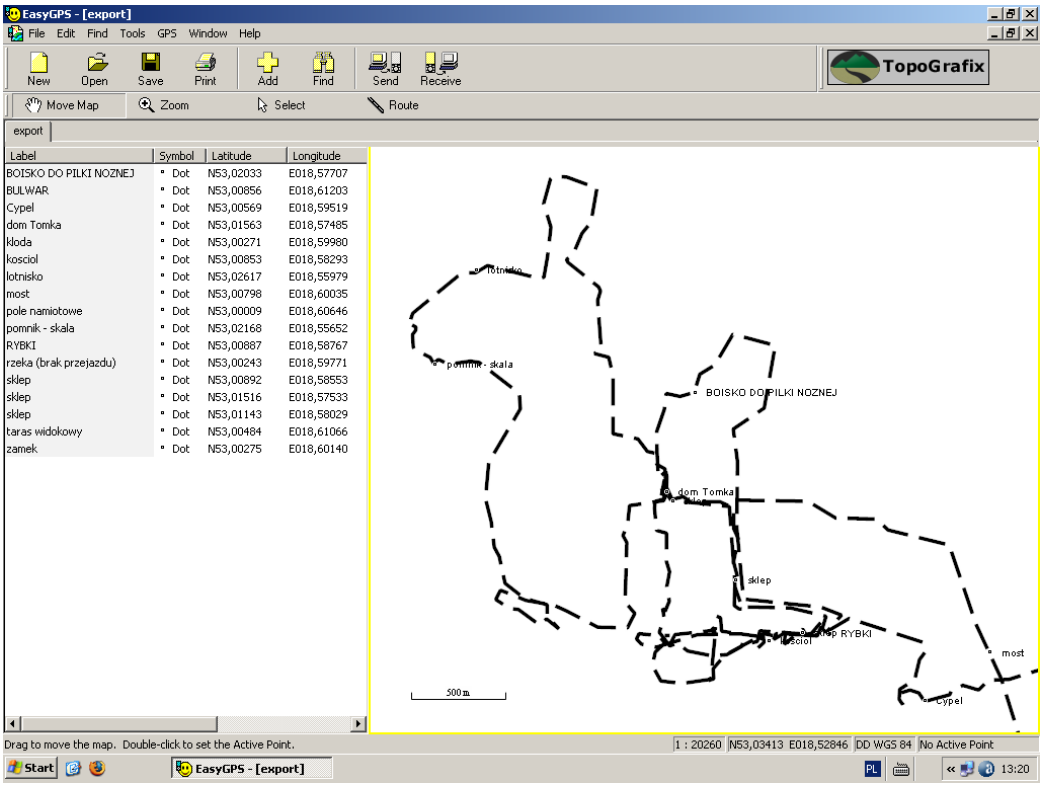
- *clear()* - czyści cały obiekt

- *addAttribute()* - dodaje nowy atrybut, pobierając jako argumenty nazwę atrybutu i jego wartość

Tak stworzone dane wprowadzane są do strumienia bajtów i przekazywane jako wynik.

7.5.1 GPX

GPX jest standardem schematu XML, przeznaczonym do wymiany danych geograficznych. Pozwala na opisywanie punktów nawigacyjnych, ścieżek i tras. Przechowuje współrzędne geograficzne punktów oraz, dla wyróżnionych miejsc, dokładniejsze opisy. Ze względu na przenośność tego formatu *Nabuchodonozor* posiada klasy i metody, obsługujące parsowanie plików GPX i eksport do nich.



Rysunek 7.3: Ścieżki w formacie GPX otworzone w programnie EasyGPS

Użytkownik *Semiramidy* może więc bezproblemowo wprowadzać do bazy ścieżki pochodzące z plików GPX, jak również uzyskać trasy z bazy danych

w tym formacie. Dzięki temu serwis internetowy *Nabuchodonozora* nie musi być ograniczony jedynie do zapisów pochodzących z modułu *Gilgamesh*, lecz bezproblemowo może służyć szerszemu gronu użytkowników, rejestrujących trasy różnymi sposobami i odczytujących je za pomocą odmiennych narzędzi.

Punkty nawigacyjne opisane są w GPX w następujący sposób:

```
<wpt lat="53.011431" lon="18.580288">
  <name>sklep</name>
  <desc>sklep</desc>
</wpt>
```

Atrybutami znacznika *wpt* są szerokość i długość geograficzna punktu, wewnątrz tej struktury mogą pojawić się nazwa i opis miejsca.

Pliki GPX są w stanie przechowywać informacje o dowolnej ilości punktów nawigacyjnych. Posiadają również opcję podziału na wiele ścieżek. Ogólny schemat definicji pojedynczej trasy wygląda następująco:

```
<trk>
  <trkseg>
    <trkpt lat="53.00875841688833" lon="18.58204280913981">
      <ele>37.0</ele>
    </trkpt>
    <trkpt lat="53.00900663563774" lon="18.579108871642156">
      <ele>37.0</ele>
    </trkpt>
     ...
  </trkseg>
   ...
</trk>
```

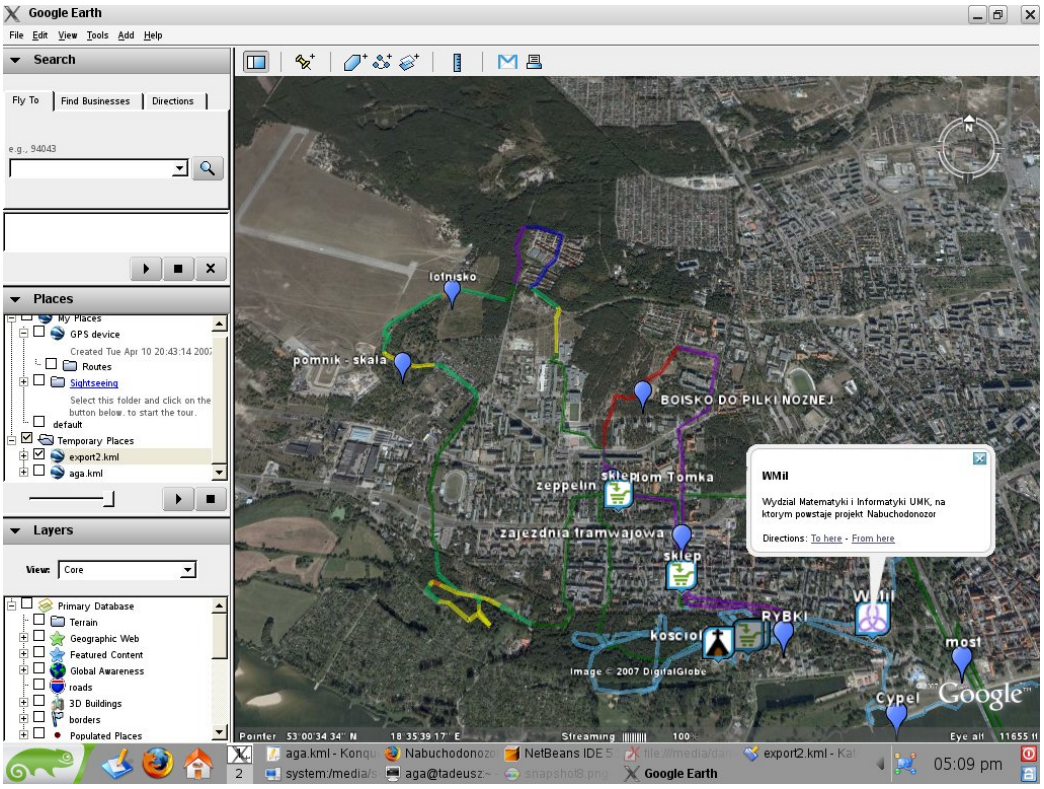
Kolejno pojawiające się znaczniki otwierają trasę, jej segment, konkretny punkt. Może się jeszcze pojawiać wysokość punktu nad poziomem morza.

Rysunek przedstawia kilka ścieżek zarejestrowanych w Toruniu przy użyciu naszej aplikacji mobilnej. Trasy te zostały wprowadzone do bazy danych i wyeksportowane do GPX. Zrzut ekranu pokazuje ich odczyt w aplikacji EasyGPS.

7.5.2 Eksport do KML

Opisane dotychczas formaty, do których można eksportować dane, służą wymianie danych geograficznych między aplikacjami oraz użyciu danych z *Nabuchodonozora* jako zastępstwa lub uzupełnienia mapy w terenie. Ostatni format jest, podobnie jak GPX, odmianą XML. Opracowany został przez firmę Keyhole, Inc, przejętą później przez Google. Odczytywany może być głównie przez programy takie jak Google Earth, Gogle Maps i Google Maps

for Mobile. Eksport ten daje możliwość nadania ścieżkom elementu trójwymiarowości, pozwala je wyświetlać na podkładzie zdjęć satelitarnych oraz skonfrontować z danymi o ciekawych obiektach dostępnymi na serwerach Google. Google Maps pozwala wybierać, jakie elementy wczytanego pliku mają być wyświetlane.



Rysunek 7.4: Ścieżki wyświetlone w programie Google Earth

Podobnie jak w pozostałych formatach, w KML-u zakodowane zostają informacje o punktach specjalnych oraz przebiegu tras. Liniom można przypisywać barwy, więc kolorowane są zgodnie z przyjętą w projekcie konwencją oznaczania nawierzchni.

Jak już zostało wspomniane, KML jest formatem typu XML. Omówione zostaną tylko te elementy języka, które zostały wykorzystane w *Marduku*. Standardowo, dokument rozpoczyna deklaracja użytego standardu:

```
<?xml version="1.0" encoding="UTF-8"?>
  <kml xmlns="http://earth.google.com/kml/2.1">
```

Dalej wprowadzone zostają definicje kolorów użytych do oznaczenia rodzajów drogi. Przykładowo:

```
<Style id="colors7">
  <LineStyle>
    <color>7f0000c0</color>
    <width>4</width>
  </LineStyle>
  <PolyStyle>
    <color>7f0000c0</color>
  </PolyStyle>
</Style>
```

Powyższy fragment definiuje ścieżkę koloru ciemnoczerwonego (odpowiadającego drodze piaszczystej). Łamana łączy punkty linią, rozpiętą nad ziemią, tworząc jednocześnie półprzezroczysty mur (stąd potrzebne dwie definicje koloru - dla linii i wypełnienia powstałych ścian).

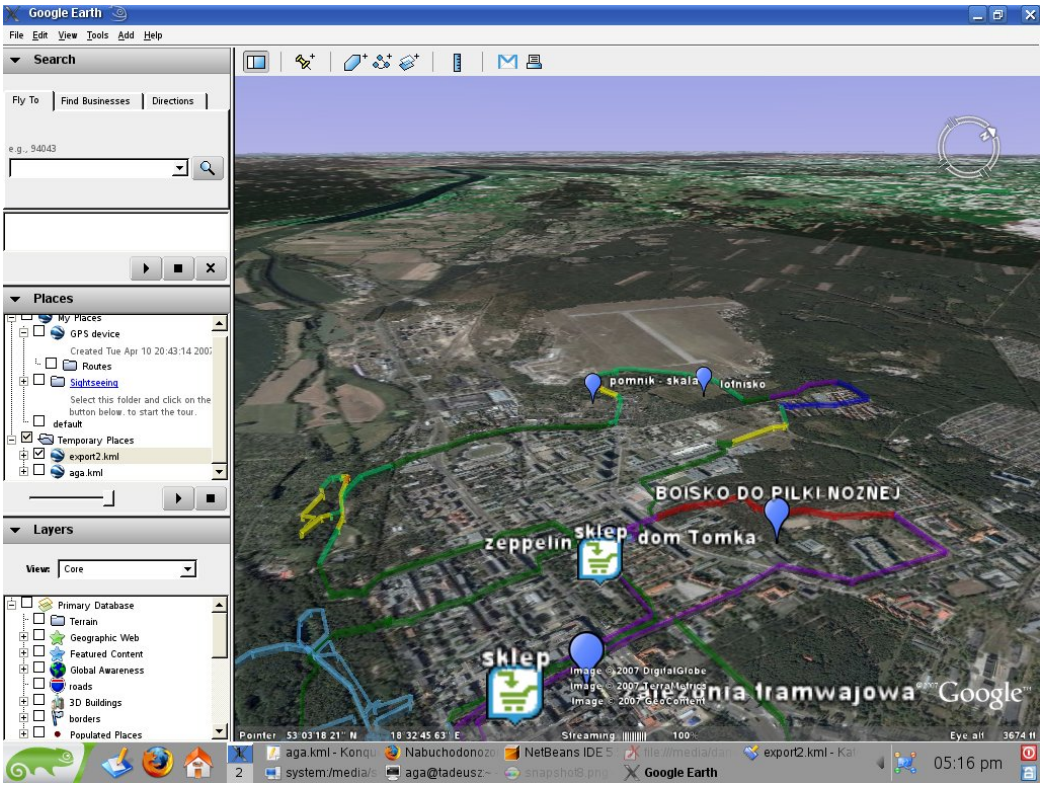
Początek dokumentu zawiera również definicje ikon, użytych do reprezentacji ważnych punktów. Jako że użycie Google Earth wymaga stałego połączenia z internetem, nic nie stoi na przeszkodzie, aby grafiki umieszczone były na serwerze i tylko w razie potrzeby ładowane. Definicja dowolnego marka ogranicza się do ustalenia jego nazwy i określenia URL-a do pliku z ikoną:

```
<Style id="mark21">
  <IconStyle>
    <Icon>
      <href>http://www.mat.umk.pl/~agniecha/images/mark21.png</href>
    </Icon>
  </IconStyle>
</Style>
```

Po definicjach używanych stylów pojawiają się najpierw punkty markowane, wraz z opisami nadanymi przez użytkowników *Gilgamesha* i *Semiramidy*. Dalej zaś pojawia się najważniejsza treść, czyli przebieg tras. Dane te mogą być dzielone na foldery, co ułatwia wybór wyświetlanych komponentów podczas odczytu pliku w Google Earth. W skrócie, dane te wyglądają jak poniżej:

```
<Folder a='aga'>
  <name>Placemarks</name>
  <Placemark>
    <name>sklep</name>
    <description>sklep</description>
    <styleUrl>#mark21</styleUrl>
    <Point>
      <coordinates>18.580288,53.011431</coordinates>
    </Point>
  </Placemark>
  ...
</Folder>
```

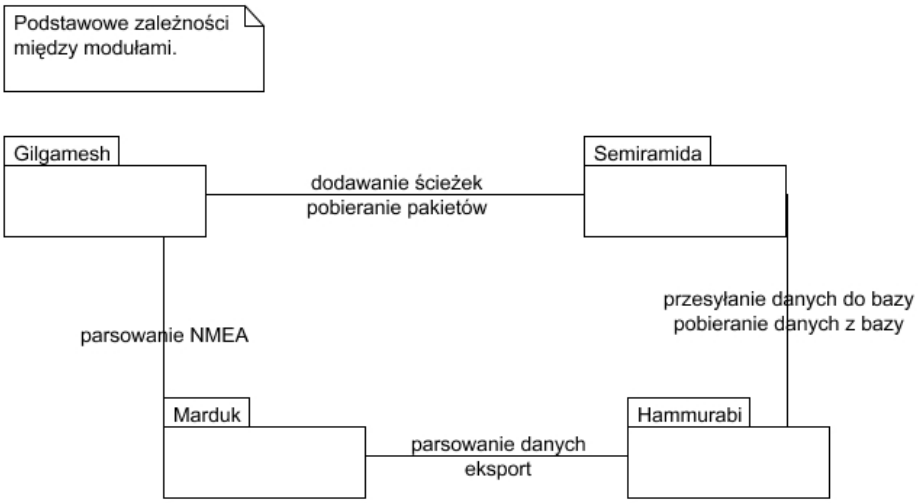
```
<Folder>
  <name>Paths</name>
  <Placemark>
    <styleUrl>#colors3</styleUrl>
    <LineString>
      <extrude>1</extrude>
      <altitudeMode>relativeToGround</altitudeMode>
      <coordinates>
        18.580155871641665,53.01121435438415,20
        18.580386387266568,53.011552198133046,20
        18.58005351226668,53.01230149500668,20
        ...
      </coordinates>
    </LineString>
  </Placemark>
  ...
</Folder>
```



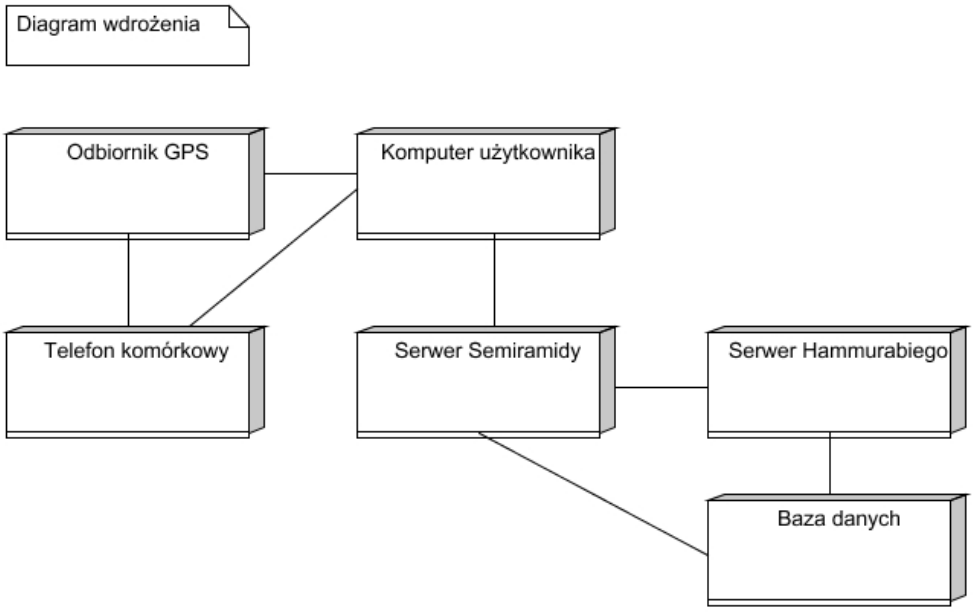
Rysunek 7.5: Ścieżki wyświetlone w programie Google Earth

Rozdział 8

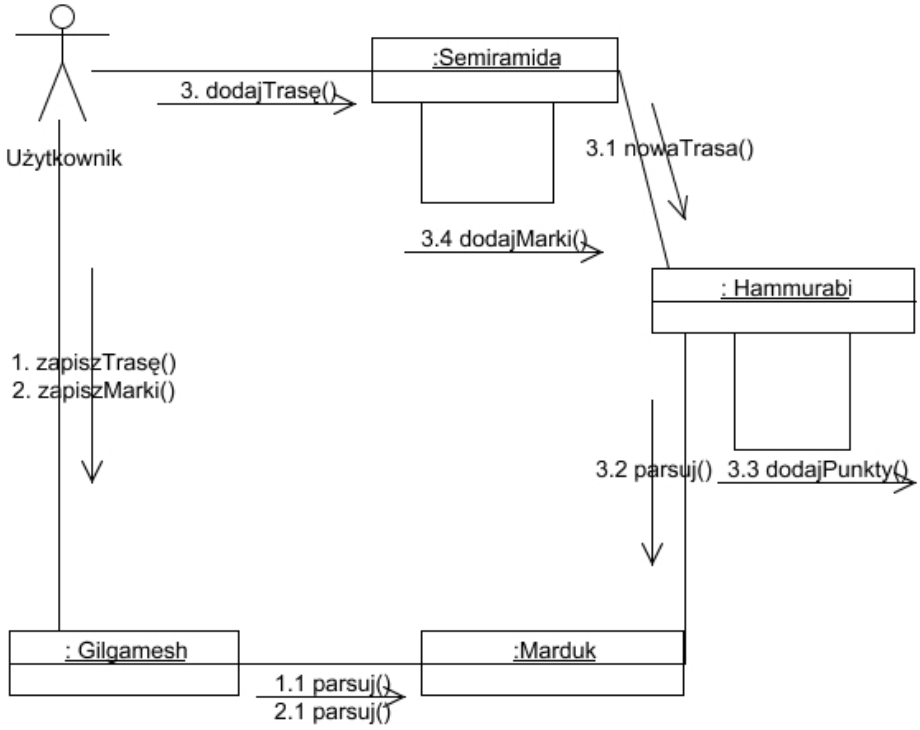
Diagramy UML



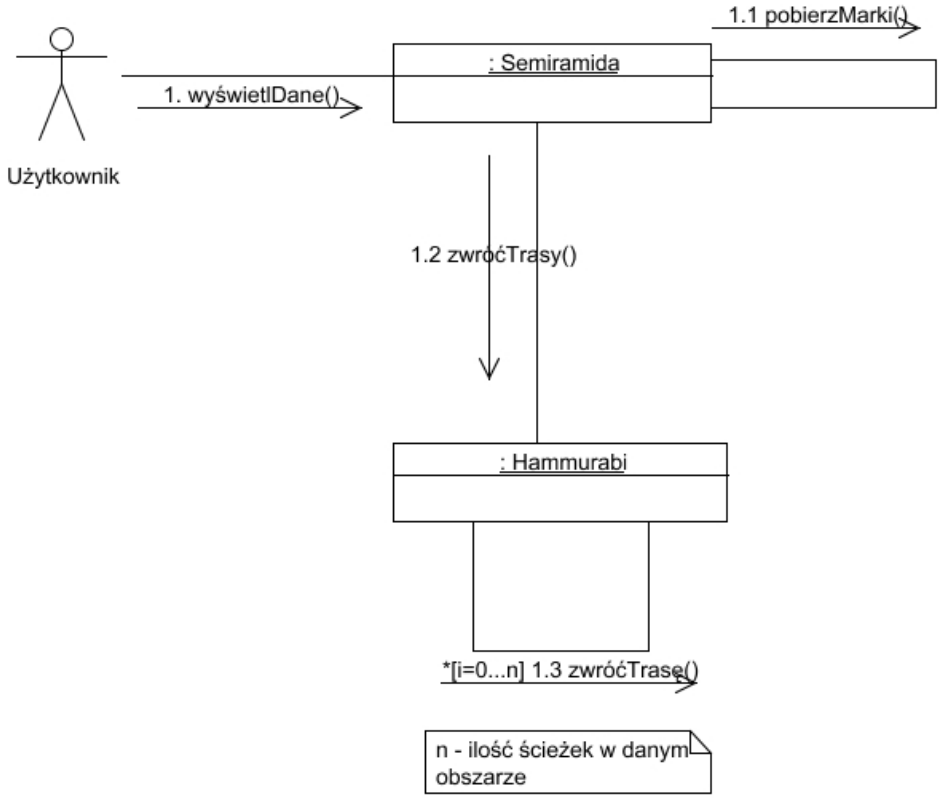
Rysunek 8.1: Diagram przedstawiający podstawowe zależności między modułami



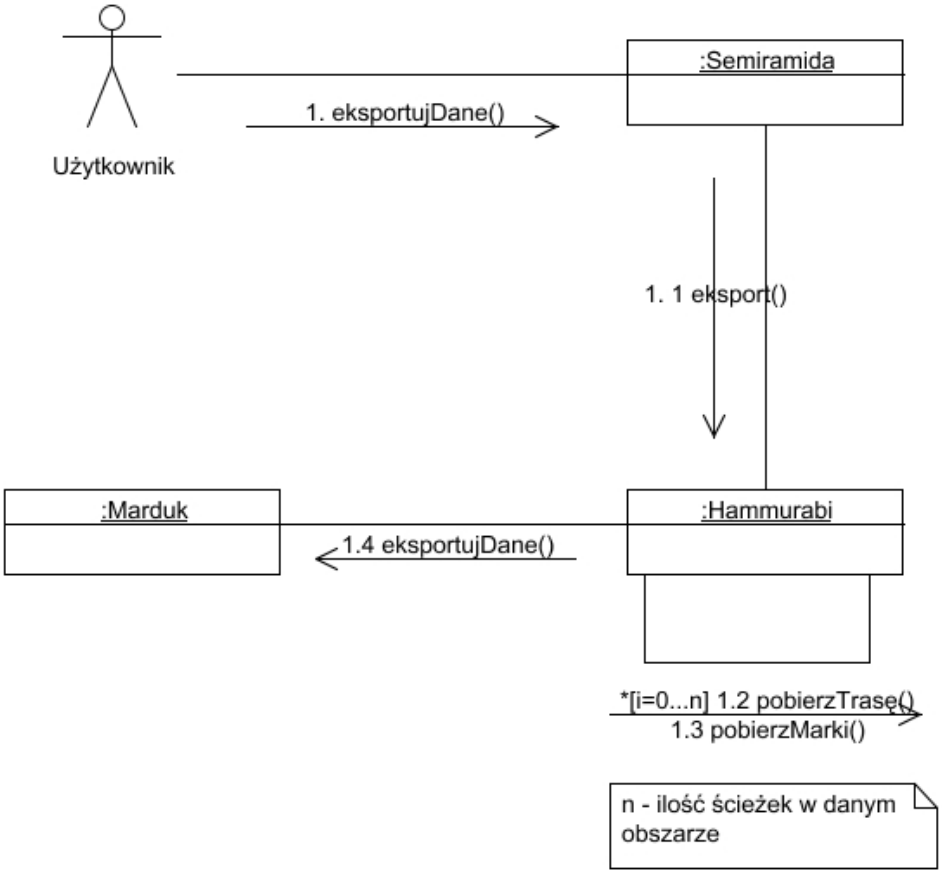
Rysunek 8.2: Wdrożenie projektu



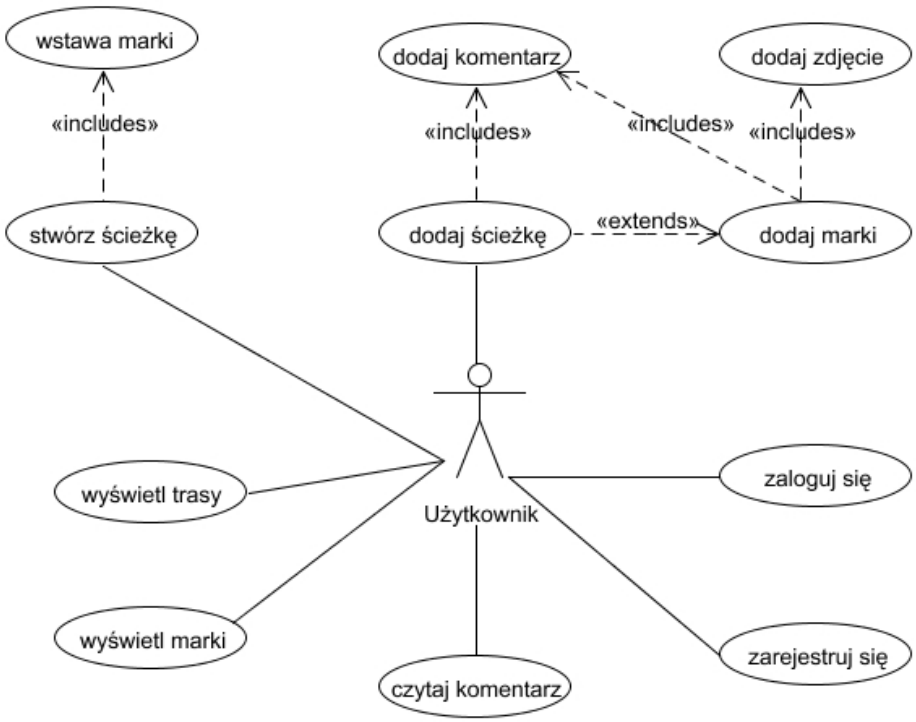
Rysunek 8.3: Diagram przedstawiający kolejne etapy dodawania ścieżki do serwisu



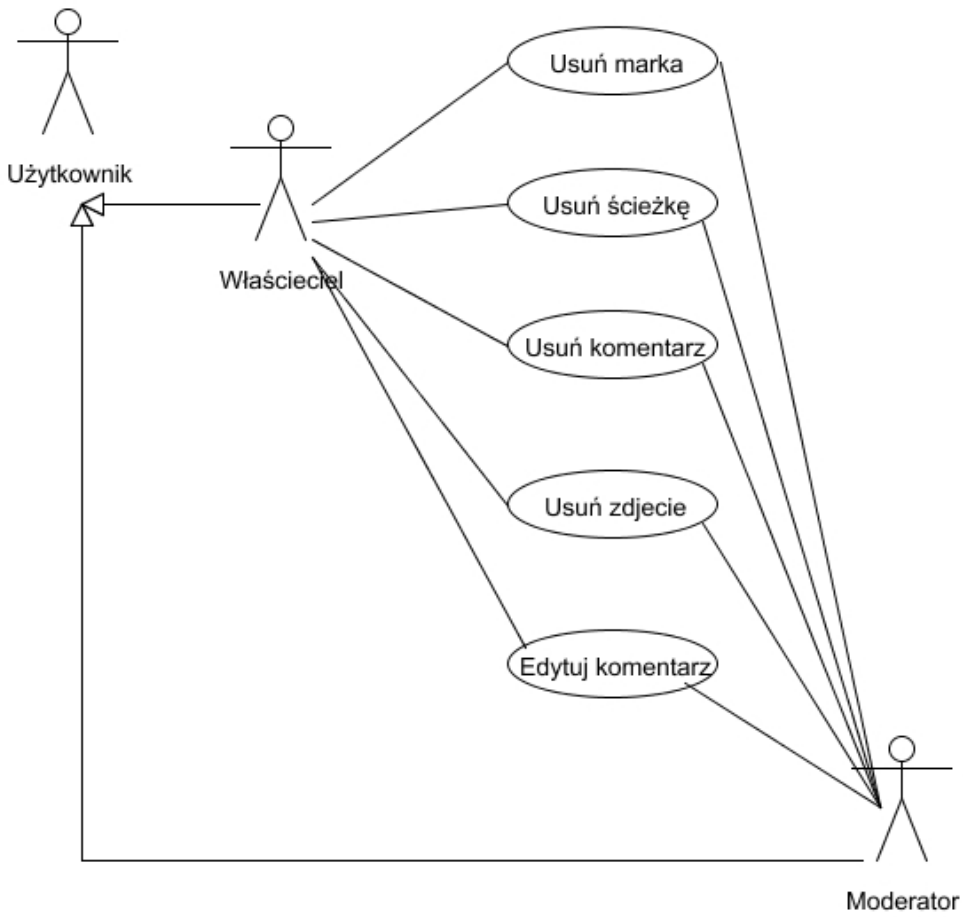
Rysunek 8.4: Pobieranie danych z serwisu



Rysunek 8.5: Eksport danych z serwisu



Rysunek 8.6: Podstawowe przypadki użycia



Rysunek 8.7: Podstawowe przypadki użycia