1. **MAIN SYSTEM**
    a. REST
        i. GET /main-system/rest/parking-places?street={string} - *Zewnętrzne API dla zewnętrznych systemów monitorowania zajętości miejsca na ulicach - pozwalające na udostepnienie podstawowych informacji innym systemom (**3d)***
        http://localhost:8080/main-system/rest/parking-places?street=Friedleina
        ii. GET /main-system/rest/reports/pdf?from={yyyy-MM-dd'T'hh:mm:ss}&to={yyyy-MM-dd'T'hh:mm:ss}&sectionNumber={Long} - zwraca raport w pdf wszystkie parametry są obowiązkowe (**3e)**
        http://localhost:8080/main-system/rest/reports/pdf?from=2015-08-04T10:11:30&to=2018-08-10T10:11:30&sectionNumber=1
        iii. POST /main-system/rest/tickets - tworzy bilet, przyjmuje JSON z polami: long placeNumber, long sectionNumber, long durationInMinutes, String registerPlate. Wykorzystywany przez **parking-meter. (1)**
        http://localhost:8080/main-system/rest/tickets

```
{
        "placeNumber": "2",
        "sectionNumber": "2",
        "durationInMinutes": "44",
        "registerPlate": "AXZ222"
}
```

    b. SOAP
        i. ParkingPlaceSensor - służy do komunikacji z **parking-place-sensor (1),** udostępnia metody, *getAllParkingPlaces()* - zwraca wszystkie miejsca*, markParkingPlace()* - zmienia stan miejsca (OCCUPIED, FREE) - WDSL: /main-system/ParkingPlaceSensorImpl?wsdl
        ii. ParkingPlaceService - *udostępnia metode getParkingPlaces(), która pobiera listę miejsc parkingowych na podstawie ulicy **(3d)***
    c. EVENTS
        i. Metoda detectExpiredOrNotBought() w beanie EventService sprawdza co pewien interwal czasowy (domyślnie 5) czy nie wystąpiła jedna z dwóch sytuacji: bilet wygasł a miejsce jest zajęte, lub miejsce zajęte i bilet nie został wogóle kupiony. Dodatkowo, gdy bilet zostaje kupioiny lub stan miejsca zmieniony również wysyłamy wiadomość do kolejki. **(3b).**
2. **DASHBOARD**
    a. przygotowanie Dashboard wizualizującego informacje o stanie zajętość miejsc parkingowych. Użytkownik z rolą **WORKER** widzi tylko miejsca w jego strefie. Użytkownik z rolą **ADMIN** widzi wszystko. Dodatkowo mamy i18n (Polski i angielski) . (**3a)**
    http://localhost:8080/mobile-notificator/rest/notifications?from=2015-08-04T10:11:30&section=1
3. **MOBILE-NOTIFICATOR**
    a. Pobiera notyfikacje z kolejki eventów i je persystuje. **(3c)**
    b. REST
        i. GET /mobile-notificator/rest/notifications?from={yyyy-MM-dd'T'hh:mm:ss}&sectionNumber={Long} - *System alertów dla kontrolerów strefy. Wykorzystując JMS wysłać dane do modułu wystawiającego powiadomienia dla urządzeń mobilnych – Powiadomienia są wystawiane za pomocą API REST zasilanego danymi za pomocą JMS.* **(3c)**
        http://localhost:8080/mobile-notificator/rest/notifications?from=2015-08-04T10:11:30&section=1
4. **PARKING-METER (2)**

    a. Możemy kupić bilet na określone miejsce na określoną liczbę minut. Komunikujemy się z **main-system** za pomocą REST.
5. **PARKING-PLACE-SENSOR (1)**
    a. Możemy zasymulować zmiane stanu miejsca. FREE or OCCUPIED. Komunikoujemy się z modułem **main-system** za pomocą **SOAP.**

Jak odpalić? ($HOME_PROJECT -** korzeń projektu, **$HOME_WILDFLY - katalog domowy wildfly**)**

1. Idź do $HOME_WILDFLY/standalone/configuration i wykonaj kopie pliku standalone-full.xml
2. Skopiuj plik $HOME_PROJEKT/standalone-full.xml do $HOME_WILDFLY/standalone/configuration
3. Idź do $HOME_WILDFLY/bin i uruchom *./standalone.sh -c standalone-full.xml -DANTLR_USE_DIRECT_CLASS_LOADING=true,* wyświetlą się błędy
4. Musisz zainstalować postgresa, ustaw hasło dla postgres - MJordan23
5. Utwórz jako użytkownik postgres baze wildfly
6. Zdeplojuj driver postgresa z $HOME_PROJECT na serwerze z taką smą nazwą jak plik
7. Uruchom ponownie serwer
8. Wejdź w $HOME_PROJECT i odpal mvn clean install - powinno się wszystko zdiplojować

Użyte technologie

1. Provider JPA - Hibernate
2. PDF - iText
3. Lombok