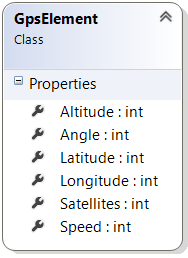
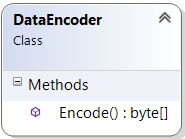
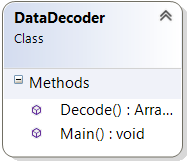
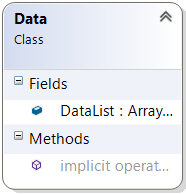
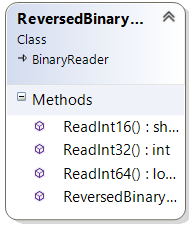
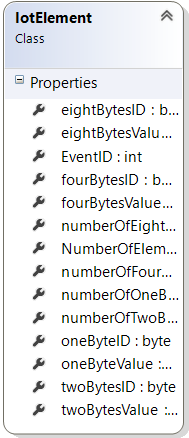
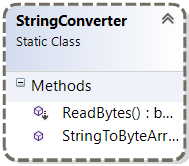
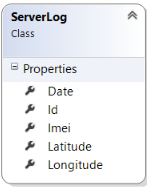
Pirma užduotis. Siunčiamų duomenų dekodavimas/užkodavimas.

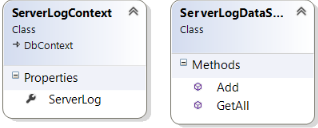


Projektas sudarytas iš 7 klasių. Klasės GpsElement, IotElement ir Data yra duomenų struktūros, kurios neturi jokių metodų. Klasę StringConverter turi 2 metodus: ReadBytes() skirtas perskaityti konkretų baitų kiekį, o StringToByteArray būtų galima panaudoti klasės veikimo patikrinimui(nes šiuo metu testų dar nėra). Klasė ReversedBinaryreader skirta baitų nuskaitymui iš little endian į big endian. DataDecoder klasėje yra tuščias Main() metodas(kurio galėtų ir nebūti). Kitas metodas tai Decode(). Tai labai kompleksinis metodas, kurio paskirtis: dekoduoti jam siunčiamus duomenis. Šis metodas priima baitų baitų masyvą ir gražiną sąrašą dekoduotų elementų. Klasėje DataEncoder yra metodas Encode(), kuris priima sąrašą elementų ir gražina baitų masyvą.

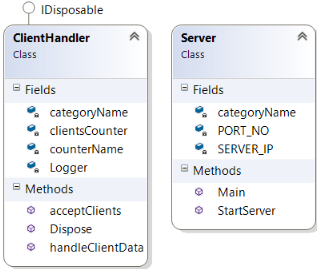
2 užduotis. Serverio ir klientų komunikacija.



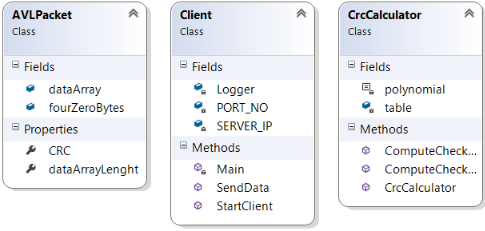
Pradėsiu nuo duomenų bazės, kuri skirta duomenų saugojimui. Projekte Server.Domain yra duomenų struktūros tipo klasė ServerLog. Naudojant „Code first“ metodą buvo sukurta lentelė duomenų bazėje būtent su tokio tipo stulpeliais, kokie yra klasės ServerLog „*properčiai*“.



Projektas Server.DataAccess taip pat skirtas duomenų bazės kūrimui ir duomenų manipuliavimui. Klasė ServerLogContext paveldi informaciją iš DbContext, kuri ir yra skirta duomenų bazės sukūrimui pagal pateiktą ServerLog modelį. O klasė ServerLogDataService skirta duomenų manipuliavimui. Metodas Add() skirtas duomenų pridėjimui į duomenų baze. jame sukurtas funkcionalumas: pakeitimas iškart išsaugojami duomenų bazėje ir tikrinama ar tokie duomenis duomenų bazėje neegzistuoja. Jei egzistuoja, tada duomenys nėra saugojami. GetAll() funkcija sudeda visus duomenis į sąrašą ir jį gražina.



Serveris turi klasę Server, kuri atsakinga už serverio paleidimą. Main() metode iš pradžių tikrinama ar yra sukurtas „Performance Counter“, kuris atsakingas už klientų skaičiavimą per *perform*. Tada kviečiamas metodas StartServer(), kuris sukuria TcpListener objektą, pasiunčia šį objektą į ClientHandler klasės metodą acceptClients(). Šis metodas nepertraukiamai priiminėja klientus tol, kol serveris įjungtas. Priimant klientą sukuriamas naujas TcpClient objektas, kuris toliau paduodamas į handleClientData užduotį. Šis metodas naudodamas NetworkStream priima kliento siunčiamus duomenis ir asinhcroniškai juos apdoroja. Priimami visi duomenys, tada patikrinama ar sutampa gautų duomenų ir kliento išsiustų duomenų CRC polinomas. Jei nesutampa, tada duomenys bandoma atsiųsti iš naujo. Tada naudojantis anksčiau aptarta Codec8 biblioteka dekoduojamas baitų masyvas ir iš sąrašo duomenų bazėje išsaugoma data, ilguma bei platuma. Taip pat viskas yra *loginama* į konsolę, bei į failą.



Projektas Client turi klasę Client, kuri atsakinga už klientų sukūrimą. Main() metode pasirenkama, kiek klientų norite sukurti. Tada kviečiamas metodas StartClient(), kuriame sukuriamas TcpClient kliento objektas, kuris prisijungs prie serverio ip adreso ir porto. Šis objektas tada siunčiamas į užduotį SendData(). Šis metodas naudodamas NetwrokStream siųs duomenis serveriui, kuris juos priima. Kliento duomenys visada yra tie patys ir aprašyti klasėje AVLPacket. Pagrindinis duomenų masyvas gaunamas iš Codec8 projekto klasės StringConverter funkcijos StringToByteArray(). Kliento duomenys taip pat yra įrašomi į konsolę ir failą naudojantis log4net biblioteka. CrcCalculator klasės metodai skaičiuoja polinomą, kuris skirtas sužinoti ar siunčiami ir atsiųsti duomenys yra tokie pat. Taip pat tiek serverio tiek kliento ip adresas ir portas yra imamas iš projekto app.cofig failo.