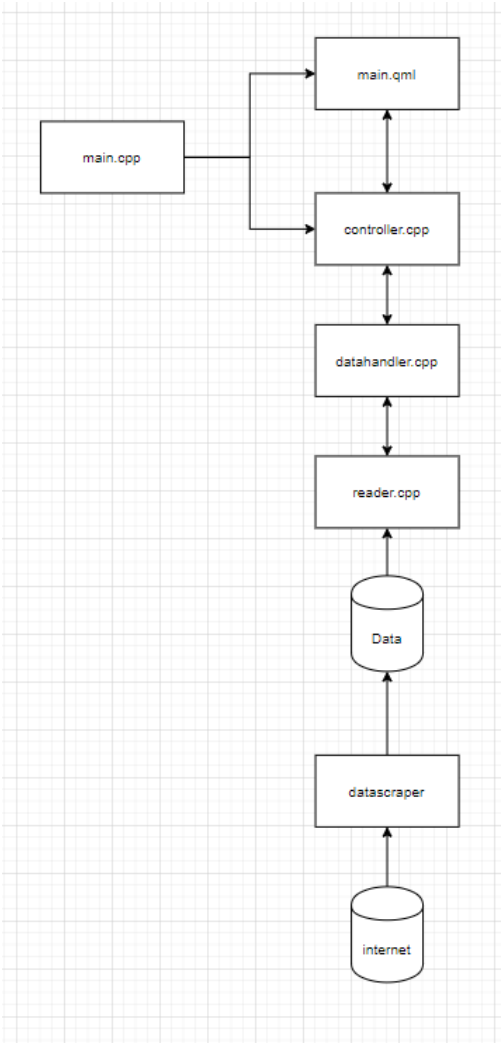
OhSu Dokumentaatio

Valtteri Huhdankoski
Joonas Puumala
Elias Örmä
Anttoni Tukia

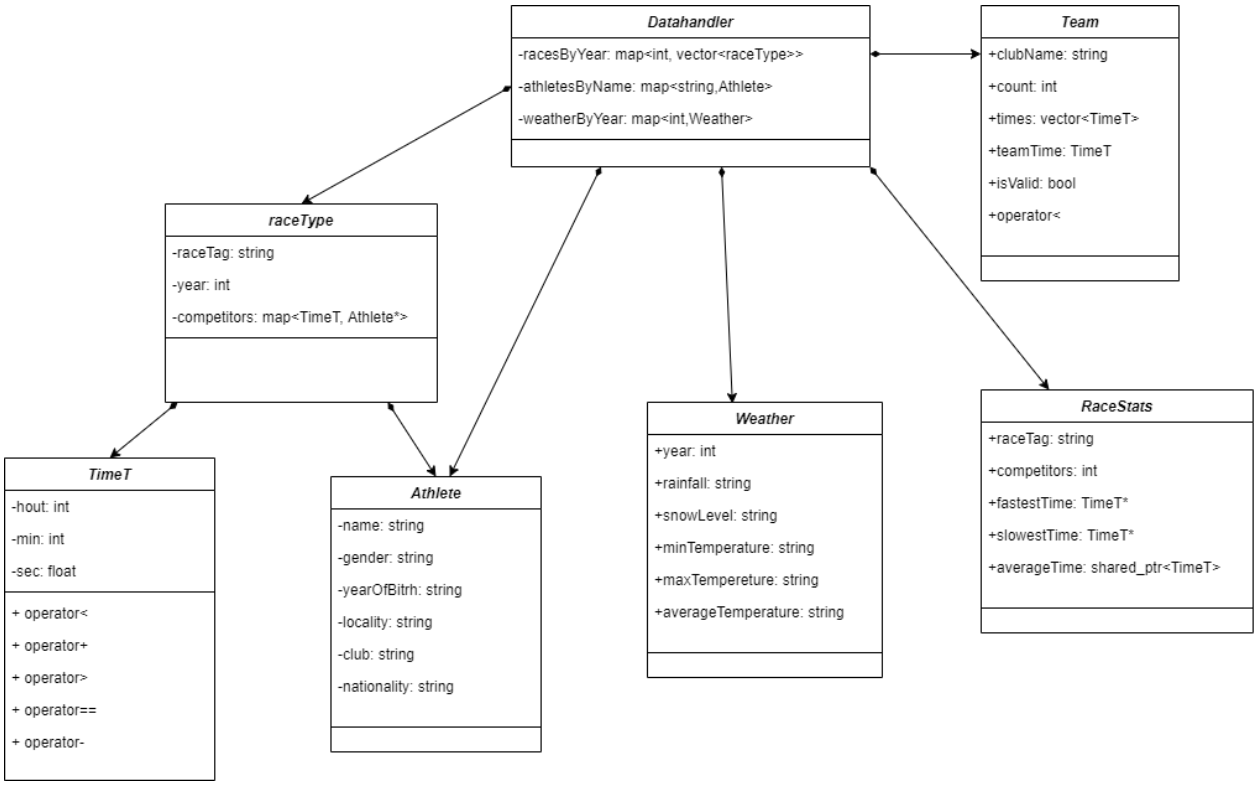
Korkean tason kuvaus

Ohjelman GUI ja toiminnallisuus on toteutettu käyttämällä C++ 11 ja QML 14.2 teknologioita. Ohjelman datascraper osio on toteutettu käyttämällä Pythonia. Pythonissa on käytetty kolmannen osapuolen selenium ja hashlib kirjastoa.



Kuva 1. Ohjelman rakenne

Data kerätään Finlandiahiihdon nettisivuilta ja tallennetaan sopiviin tekstitiedostoihin. Tekstitiedostoista data luetaan ja muutetaan oliomuotoon. Datahandler omistaa kaiken tarpeellisen datan, jotta haut saadaan toteutettua. Kuvassa 2 on esitetty dataluokkien vastuualueita.



Kuva 2. Ohjelmiston tietorakenteet

Datan tallennukseen ohjelmassa on luokkia eri tarpeisiin. TimeT, Athlete ja RaceType ovat yleisen tason luokkia, joita tarvitaan lähes jokaisessa haussa. Lisäksi koodin selventämiseksi ja haluttujen tietojen etsimiseksi ohjelmaan on toteutettu luokat RaceStats, Team ja Weather, jotka edustavat datahandlerin metodien paluuarvoja ja pitävät kaiken halutun tiedon yhden olion sisällä.

TimeT: Ajan säilöntään ja vertailuja varten on toteutettu luokka, joka sisältää tiedon kilpailijoiden ajoista.

Athlete: Yksittäistä urheilijaa kuvataan Athlete-luokalla. Sen tiedossa on yksilöivä id, nimi, seura, kansallisuus, sukupuoli, syntymäaika ja paikkakunta.

RaceType: Yksittäistä kisaa kuvaava luokka, joka tietää kisan aikajärjestyksen ja tulosten hiihtäjät.

DataHandler: Suorittaa varsinaiset toiminnot datalle yllä olevien luokkien avulla.

Lisäksi datan käsittelyä varten on luokat:

RaceStats: Käytetään varastoimaan yksittäisen kisan tietoja, joita erityisesti statsByEachYear() -metodi käyttää.

Team: Käytetään varastoimaan joukkueen (4 urheilijaa) tietoja ja mm. yhteisaikaa.

Weather: Käytetään varastoimaan tietoja yksittäisen vuoden sääolosuhteista.

Controller

Controller luokka on vastuussa Front endin toiminnallisuudesta. Tähän kuuluu: inputtien error check, DataHandlerin haku luokkien kutsuminen, sekä hakudatan palauttaminen Front endille.

vector<QString> getEveryYear()	Palauttaa kaikki datassa esiintyvät vuodet
Void search(QString state, QString year1, QString year2, QString starttime, QString endtime, QString racetag1, QString racetag2, QString athletename, QString sizeofranking, QString gender);	Kutsuu datahandlerin metodeita riippuen hakutyypistä
Void setNewRaceTags(int searchType, QString year1, QString year2)	Asettaa uudet racetagit UI ikkunaan riippuen asetetuista vuosista.
TimeT convertToTimeT(QString str)	Apufunktio stringin muuttamiseksi TimeT:ksi
Void logError(std::string msg)	Apufunktio erroreille
Void callShowResultsInQML()	Lähettää signaalina datahaun tulokset frontendille (main.qml)

DataTypeConverter

Tämä luokka muuttaa datahandlerluokan muodostaman datan oikeanlaiseen tietorakenteeseen riippuen Frontend luokkien tietorakenne tarpeista. Tämän luokan takia Ohjelman jatkokehittäminen helpottuu, koska datahandleriin tehdyt uudet toteutukset voivat olla ”Järkevässä” paluuarvoluokassa. Myös uusien datan näyttöluokkien muodostaminen tulevaisuudessa helpottuu, koska tämä luokka kätkee sisäänsä datatyyppimuunnoksen.

QVector<QVector<QString>> readByTime(int year, TimeT startTime, TimeT endTime, string raceTag);	Muokkaa paluuarvon sopivaksi Frontendille
QVector<QVector<QString>> compareResultByYear(string raceTag,int Year1, int Year2, int sizeOfList);	Muokkaa paluuarvon sopivaksi Frontendille
QVector<QVector<QString>> compareResultByYearNames (string raceTag,int Year1, int Year2, vector<string> names)	Muokkaa paluuarvon sopivaksi Frontendille
QVector<QVector<QString>> compareDistanceByYear(string raceTag1, string raceTag2, int year);	Muokkaa paluuarvon sopivaksi Frontendille
QVector<QVector<QString>> statsByEachYear();	Muokkaa paluuarvon sopivaksi Frontendille
QVector<QVector<QString>> athleteTimeDevelopment(string athlete, int startYear, int endYear);	Muokkaa paluuarvon sopivaksi Frontendille
QVector<QVector<QString>> averageSpeedByRanking(int startYear, int endYear, int sizeOfList, string raceTag);	Muokkaa paluuarvon sopivaksi Frontendille
QVector<QVector<QString>> bestAthleteByGenderInPeriod(string gender, int startYear, int endYear, string raceTag);	Muokkaa paluuarvon sopivaksi Frontendille
QVector<QVector<QString>> sortedResultsByClubName(int year)	Muokkaa paluuarvon sopivaksi Frontendille
QVector<QVector<QString>> athleteDistributionByCountry();	Muokkaa paluuarvon sopivaksi Frontendille
QVector<QVector<QString>> tenBestTeams(int year, string raceTag);	Muokkaa paluuarvon sopivaksi Frontendille

DataHandler

Controller käyttää datahandleria haluttujen tietojen etsimiseen ja jäsentelyyn, jotta data voidaan esittää käyttäjälle viewin kautta. Finlandiahiihdon tulokset on tallennettuna datahandlerin tietorakenteisiin. Yksi tietorakenne kisoille, jotka ovat lajiteltuna vuoden mukaan. Toinen tietorakenne, joka sisältää yksittäiset urheilijat. Datahandlerilla on tiedossa myös säätiedot vuosittain. Alla on listattuna datahandlerin julkisen rajapinnan metodit kuvauksineen.

map<TimeT,string> readByTime(int year, TimeT startTime, TimeT endTime, string raceTag);	Palauttaa tietyn aikaikkunan sisällä maaliin tulleiden hiihtäjien ajat ja nimet. Haku tehdään tietyn vuoden tietylle kisalle. Palauttaa tyhjän mapin, jos vuotta tai kisaa ei löydy. Palauttaa tyhjän myös, jos annettujen aikojen välillä ei ole yhtään tulosta.
map<int, vector<TimeT>> compareResultByYear(string raceTag,int Year1, int Year2, int sizeOfList);	Palauttaa tiedot kahdelta eri vuodelta saman kisan osalta. Haku tehdään tietylle määrälle hiihtäjiä, jotka ovat tulleet maaliin alkaen ensimmäisestä. Jos jompaa kumpaa vuotta ei löydy, palautetaan tyhjä map. Jos kisaa ei löydy, vuoden kohdalle sijoitetaan tyhjä vectori.
map<int, multimap<TimeT, string>> compareResultByYearNames (string raceTag,int Year1, int Year2, vector<string> names)	Palauttaa tiedot kahdelta eri vuodelta saman kisan osalta. Haku tehdään annettujen nimien ja kisan perusteella. Jos jompaa kumpaa vuotta ei löydy, palautetaan tyhjä map. Jos kilpailija ei ole osallistunut kisaan tiettynä vuonna, ajaksi annetaan NO_TIME.
map<string, vector<TimeT>> compareDistanceByYear(string raceTag1, string raceTag2, int year);	Palauttaa saman vuoden kahden eri kisan maaliintuloajat. Palauttaa tyhjän mapin, jos vuotta ei löydy. Jättää vectorin tyhjäksi, jos kisaa ei löydy.
map<int, vector<RaceStats>> statsByEachYear();	Palauttaa jokaisen vuoden kisoista seuraavat tiedot: kilpailijoiden lukumäärä, voittoaika, viimeisen aika ja keskiarvoaika
map<int, vector<raceResult>> athleteTimeDevelopment(string athlete, int startYear, int endYear);	Palauttaa tietyn urheilijan kisatiedot annettujen vuosien väliltä vuosi vuodelta. Palauttaa tyhjän mapin, jos urheilijaa ei löydy.
map<int, TimeT> averageSpeedByRanking(int startYear, int endYear, int sizeOfList, string raceTag);	Palauttaa keskivertoajan tietylle kisalle annettujen vuosien väliltä. Hakua rajataan ottamalla huomioon vain tietty määrä urheilijoita ensimmäisestä alkaen.
shared_ptr<Athlete> bestAthleteByGenderInPeriod(string gender, int startYear, int endYear, string raceTag);	Palauttaa urheilijan, joka on ollut nopein tiettyjen vuosien välillä tietyssä kisassa. Haetun urheilijan sukupuoli tulee määrittää. Palauttaa nullptr, jos urheilijaa ei löydy.
map<string, vector<competitor *>> sortedResultsByClubName(int year)	Palauttaa tietyn vuoden kilpailijatiedot aakkosjärjestyksessä kerhon nimen mukaan.
map<string,int> athleteDistributionByCountry();	Palauttaa kisoihin osallistuneiden kilpailijoiden lukumärän kansallisuuden mukaan.
map<string, list<Team>> tenBestTeams(int year, string raceTag);	Palauttaa kymmenen parasta joukkuetta annettuna vuonna. Jos tietty kisa on määritetty, palautetaan vain sen tiedot. Jos kisaa ei ole annettu, palautetaan kaikkien kisojen tiedot annetulta vuodelta. Joukkue muodostetaan neljästä saman seuran urheilijasta.
map<int, vector<string>> raceTagsYearly();	Palauttaa eri vuosina olleiden kisojen tunnukset.
Weather getWeather(int year);	Palauttaa halutun vuoden säätiedot oliona.

Reader

Reader-luokan tehtävänä on lukea Datascraper:n luoma tekstitiedosto ja tallettaa tieto haluttuihin tietorakenteisiin. Reader:n käyttä tapahtuu siten, että Datahandler luo rakentajassaan Reader-olio, ja kutsuu sen readAtheletes ja readRaces metodeja, jotka palauttavat Datahandler:lle tiedot kilpailijoista ja kilpailuista.

std::map<std::string, Athlete> readAthletes()	Lukee tiedoston ” skiDataByYearAndRoutes.txt”, luo sen pohjalta Athlete-olioita ja tallentaa ne map:iin (avaimena nimi).
std::map<int, std::vector<raceType> > readReaces	Lukee tiedoston ” skiDataByYearAndRoutes.txt”, luo sen pohjalta raceType-olioita ja tallentaa ne vector:iin map:n sisälle (avaimena vuosi).
std::vector<std::string> split	Apufunktio, joka käytetään merkkijonojen pilkkomiseen.

Datascraper

Datascraper on pythonilla kirjoitettu seleniumpohjainen webcrawler, joka kerää datan BeautifulSoup4 avulla. Seleniumilla navigoidaan sivulle ja asetetaan parametrit, jonka jälkeen bs4 saa kaiken datan suoraan HTML- tiedostosta kerralla. Ohjelma on kasattu niin että sen voi suorittaa osissa, jotta virheiden riski minimoidaan. Ohjelma muuttaa yearIndexOnMenu arvoa jokaisella suorituskerralla, jolloin scraper seuraavan kerran käynnistettäessä navigoi seuraavaan vuote en.

Ohjelma palauttaa kaikkien kisojen datat pakattuna vuosittain, jossa on tulokset merkitty kisoittain parhaasta tuloksesta huonoimpaan. Tulokset ovat CSV muodossa:

Vuodet omalla rivillä ja merkattu ~-merkillä.

Vuosien välissä kisat eritelty #-merkilä, jota seuraa kisan tiedot.

Kilpailutulokset eritelty kisojen alle seuraavasti:

```
Indeksijärjestys: ID, Kisa, aika, sija, sija miehet, sija naiset, sukupuoli,
nimi(sukunimi ja etunimi), kotipaikka, kansalaisuus, syntymävuosi, joukkue
```

Kilpailutulokset eroteltu riveillä, ja indeksit puolipisteillä.

Datascraperin toisessa osassa lisätään kaikki urheilijat erilliseen tiedostoon, jotta henkilöiden erittely sujuu helpommin datahandlerissa.

Urheilijat kirjataan CSV- muodossa, järjestyksessä:

MD5 generoitu ID, sukupuoli, nimi, kotipaikkakunta, kansalaisuus, syntymävuosi, urheilujoukkue.

Weather scraper

Lisäosana ohjelmaan lisättiin säätilan scrapeus. Etsimme erikseen tiedot siitä, milloin kisat oli järjestetty vuosittain. Niiden tietojen pohjalta querytettiin pythonin urllib3.request, ja bs4 kirjastojen avulla kaikkien kisapäivien data xml muodossa.

Tiedostoista parsittiin vuosittain koko kisojen keston ajalta säädata, ja niiden keskiarvot lisättiin vastaavan vuoden kohdalle erilliseen tiedostoon CSV muodossa.

Joka vuodelle laskettiin keskiarvot kisan ajalta: rrday:päivän sademäärä; tday:päivän keskilämpötila; snow:lumen määrä;tmin: päivän alin lämpötila;tmax: päivän ylin lämpötila

Front end-luokat

Luokkiin kuuluvat:

main.qml, sisältää ikkunan koko ohjelmalle ja funktiot, jotka ohjailevat mitä ikkunassa näytetään. Main.qml sisältää SearchView.qml ja Resultview.qml.

SearchView.qml, sisältää valikon, josta voi valita erilaisia hakumetodeja. se sisältää myös InputBox ikkunoita, RacetagComboBox.qml/ Combobox ikkunoita sekä toiminnallisuuden sille, mitkä inbox/combobox ikkunat näytetään riippuen valitusta hakumetodista.

ResultView.qml, sisältää erilaisia viewejä datalle, mitkä näytetään, kun data on noudettu tarkasteltavaksi. Tämä sisältää TableDataViewin ja GraphDataViewin.

InputBox.qml, sisältää layoutin input ikkunalle.

RaceTagComboBox.qml, sisältää comboboxin racetag datan näyttämiselle.

GraphDataView.qml sisältää graafinpiirtämiseen tarvittavan luokan

TableDataView.qml sisältää taulukkodata luokan.

Suunnitteluratkaisut

Ohjelmistossa käytimme seuraavia teknologioita: Python 3.8.2, C++ 11 ja QML 14.2. Ohjelmistossa käytetyt kirjastot sisältyvät edellä mainittuihin ohjelmointikieliversioihin. Ohjelmansuunnitteluvaiheessa käytimme MVC-mallia ohjelmiston kokonaisrakenteen suunnittelemiseen. MVC malli dominoi myös harjoitustyön tekijöiden vastuualueiden jakoa. Vastuualueet jakautuivat seuraavasti:

- Frontend: Elias, Anttoni
- Backend: Joonas, Valtteri
- DataScraper: Valtteri

Päädyimme valitsemiimme suunnitteluratkaisuihin osallistujien aikaisemman tietämyksen MVC mallista. MVC-malli oli myös sopivan epäkompleksi, joten se soveltui täydellisesti kyseisen ohjelmiston suunnittelemiseen.

Suunnittelimme ohjelman noudattamaan MVC mallia luomalla ”korkean tason” luokkakaavion heti aluksi. Jokainen näistä luokista kätki osan MVC mallin ominaisuuksista. Ohjelmiston rakenne noudattaa MVC mallia.

Datascraperin toteutukseen tarvittiin muutamia ulkoisia työkaluja. Python kirjastot selenium, BeautifulSoup4, urllib3 tarvitsi asentaa erikseen, jotta webcrawler/scraper hybridi saatiin aikaan. Seleniumilla operoitiin chromedriveria, jolla päästiin tarkasti navigoimaan sivu siihen muotoon, josta scrapeus oli mahdollista. Datan jäsentelyyn ja luettelointi järjestyi md5 hashatyillä ID:illä jonka avulla tuloksia oli helppo seurata.

Staattisen datan vuoksi se tarvitsi kerätä vain kertaluontoisesti ja tallentaa tekstitiedostoon, jonka ympärille rakensimme datan käsittelyn.

Itsearviointi

Suunnitelmamme on tukenut toteutusta hyvin. Kuitenkin uusia ominaisuuksia/luokkia on jouduttu tekemään, koska emme suunnitelleet ohjelmaa niin ”matalalla” tasolla heti aluksi. Mikään uusista luokista ja ominaisuuksista ei ole rikkonut alussa määriteltyä ohjelmiston rakennetta. Aluksi tekemämme luokkakaavio oli ainoastaan korkeantason suunnitelma.

Projektimme laatu perustuu vastuualueiden jaotteluun sekä yhteistyöhön ohjelmoijien kesken, ei niinkään alhaisen tason suunnitteluun. Toteuttamattomat toiminnallisuudet saadaan tehtyä, kunhan tiimimme pysyy alkuperäisen kaavion asettamassa luokkavastuualueissa. V arsinaisia muutoksia aikaisempaan koodiin ei todennäköisesti tarvitse tehdä, koska olemme päämäärin pysyneet samassa tehtävänannon tulkinnassa alusta asti.

Emme ole joutuneet muokkaamaan aiempaa suunnitelmaa, koska alkuperäinen suunnitelmamme oli korkean tason suunnitelma. Toiminnallisuudet olemme pystyneet toteuttamaan asettamallamme tahdilla ilman suurempia ongelmia. Emme ole myöskään joutuneet refaktoroimaan koodiamme ollenkaan (ottamatta huomioon pieniä olion sisäisiä ”lokaaleja” muutoksia) suunnitelmam me läpiviennin onnistumisen takia.


Front
+ field: type
InputErrorCheck()
List()
Histogram()
ListAndGraph()
Graph()
Controller
- dataHandler: DataHandler
+ search(QString state, QString year1, QString year2, QString starttime, QString endtime, QString racetag1, QString racetag2, QString athletename, QString sizeofranking, QString gender)
- convertToTimeT(QString str)
- logError(std::string msg)
- callShowResultsInQML()
DataHandler
- _racesByYear: std::map<int, std::vector<raceType>>
- _athletesByName: std::map<std::string, Athlete>
- _reader: Reader
+ ErrorHandling()
+ Result_by_time(int year, int start_time, int end_time)
Compare_Result_by_year(int distance, int start_year, int end_year)
+ Compare_Distance_by_year(int start_distance, int end_distance, int year)
+ Stats_by_each_year()
+ Athlete_time_development(string athlete, int start_year, int end_year)
+ Average_speed_by_ranking(int start_year, int end_year, int size_of_list)
+ Best_Athlete_by_gender_in_period(string gender, int start_year, int end_year, int size_of_list)
+ Sorted_results_by_team_name(int year)
+ Athlete_distribution_by_country()
+ Ten_best_teams()
- initializeReader()
- getTimesWithRaceTag(int year, std::string raceTag)
- getRaceStats(std::shared_ptr<raceType> singleRace)
- getRaceWithTag(int year, std::string raceTag)
- getResultsSortedByClub(raceType& race)
- updateTeam(Team& team, const TimeT& newTime, std::list<Team*>& validTeams)
- updateTeamStats(std::list<Team>& teams, std::list<Team*>& validTeams, raceType& race)
- calculateTenBest(std::list<Team*>& validTeams)
Reader
+ readAthletes()
+ readReaces(std::map<std::string, Athlete> & athleteMap)
- split(std::string & line, char delimeter)