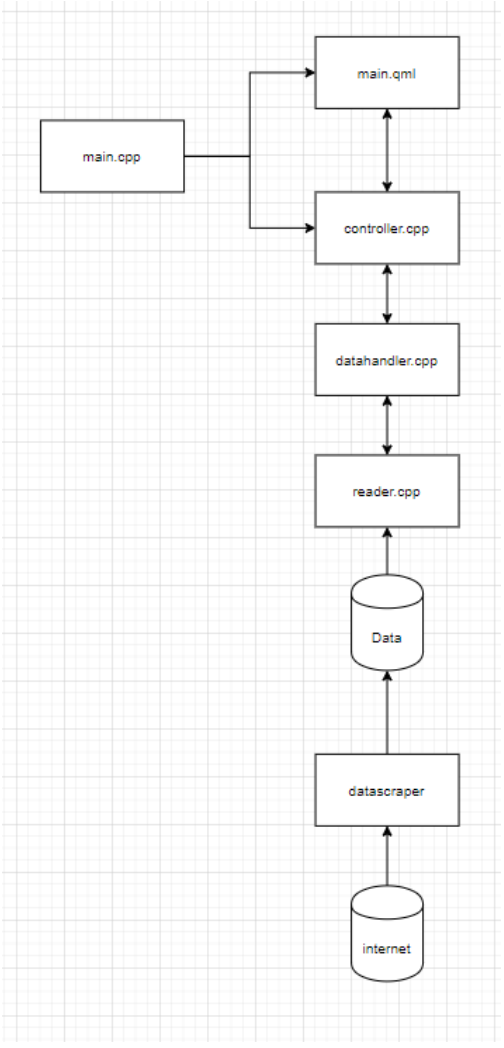
OhSu Dokumentaatio

[Tiedoston alaotsikko]

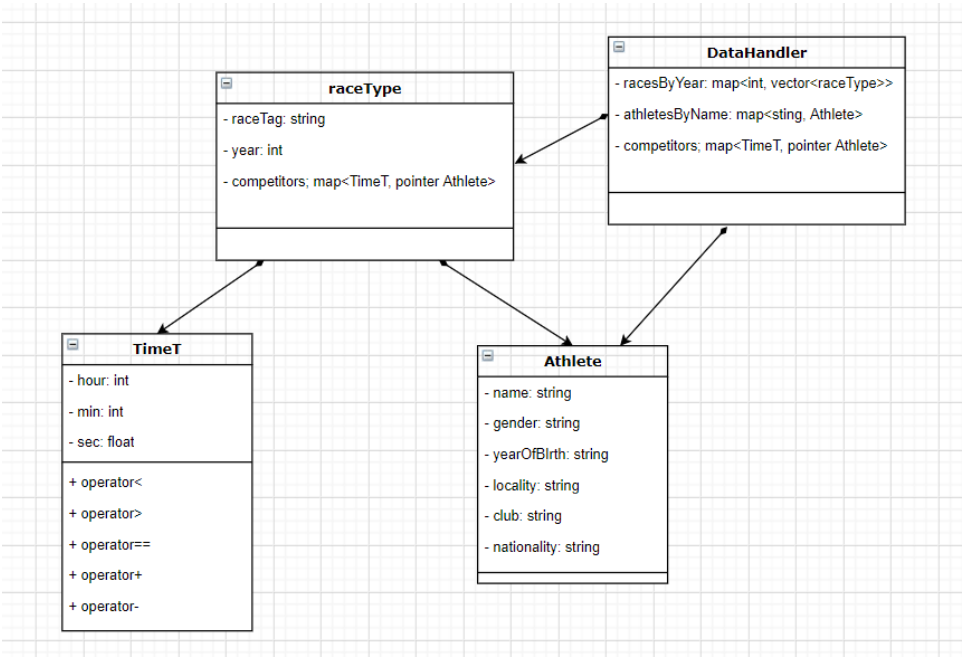
Valtteri Huhdankoski
[YRITYKSEN NIMI] [Yrityksen osoite]

Korkean tason kuvaus

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



Kuva 1. Ohjelman rakenne



Kuva 2. Ohjelmiston tietorakenteet

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.

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

Athlete: Yksittäistä urheilijaa kuvataan Athlete-luokan oliolla. 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 datahandler-luokassa on määritelty rakenteet:

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

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

Controller

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

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
TimeT convertToTimeT(QString str)	Apufunktio stringin muuttamiseksi TimeT:ksi
Void logError(std::string msg)	Apufunktio erroreille
Void callShowResultsInQML()	Signaali joka lähettää datahaun tulokset frontendille (main.qml)

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.

map<TimeT,string> readByTime(int year, TimeT startTime, TimeT endTime);	
map<int, vector<TimeT>> compareResultByYear(string raceTag,int Year1, int Year2);	
map<string, vector<TimeT>> compareDistanceByYear(string raceTag1, string raceTag2, int year);	
map<int, vector<RaceStats>> statsByEachYear();	
map<int, vector<raceResult>> athleteTimeDevelopment(string athlete, int startYear, int endYear);	
map<int, TimeT>	

averageSpeedByRanking(int startYear, int endYear, int sizeOfList, string raceTag);	
shared_ptr<Athlete> bestAthleteByGenderInPeriod(string gender, int startYear, int endYear, string raceTag);	
map<string ,map<TimeT,shared_ptr<Athlete>> sortedResultsByClubName(int year)	
map<string,int> athleteDistributionByCountry();	
map<string, list<Team>> tenBestTeams(int year, string raceTag);	

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

Datat scraperista muodossa:

Hiihtojen tulokset:

Vuodet omalla rivillä ja merkattu ~ merkillä.

Vuosien välissä kisat eritelty #, 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.
```

Front end

Luokkiin kuuluvat:

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

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

ResultView.qml, sisältää erilaisia viewejä datalle, mitkä näytetään kun data on noudettu tarkasteltavaksi

InputBox.qml, sisältää layoutin input ikkunalle.

Itsearvionti

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. Aikaisemmin 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. Varsinaisia muutoksia aikaisempaan koodiin ei todennäköisesti tarvitse tehdä, koska olemme päämäärin pysyneet samassa tehtävänannon tulkinnassa alusta asti.

Muutoksia on kuitenkin tapahtunut Front ja Controller luokkien välillä toiminnallisuuden toteuttamisesta johtuen. Datahandlerin metodejen parametrit ovat muuttuneet Frontendin vaatimusten mukaan. ANTTONI TODO READERIN MUUTOKSET VERRATTUNA ALKUPERÄISEEN UML KAAVIOON.


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)