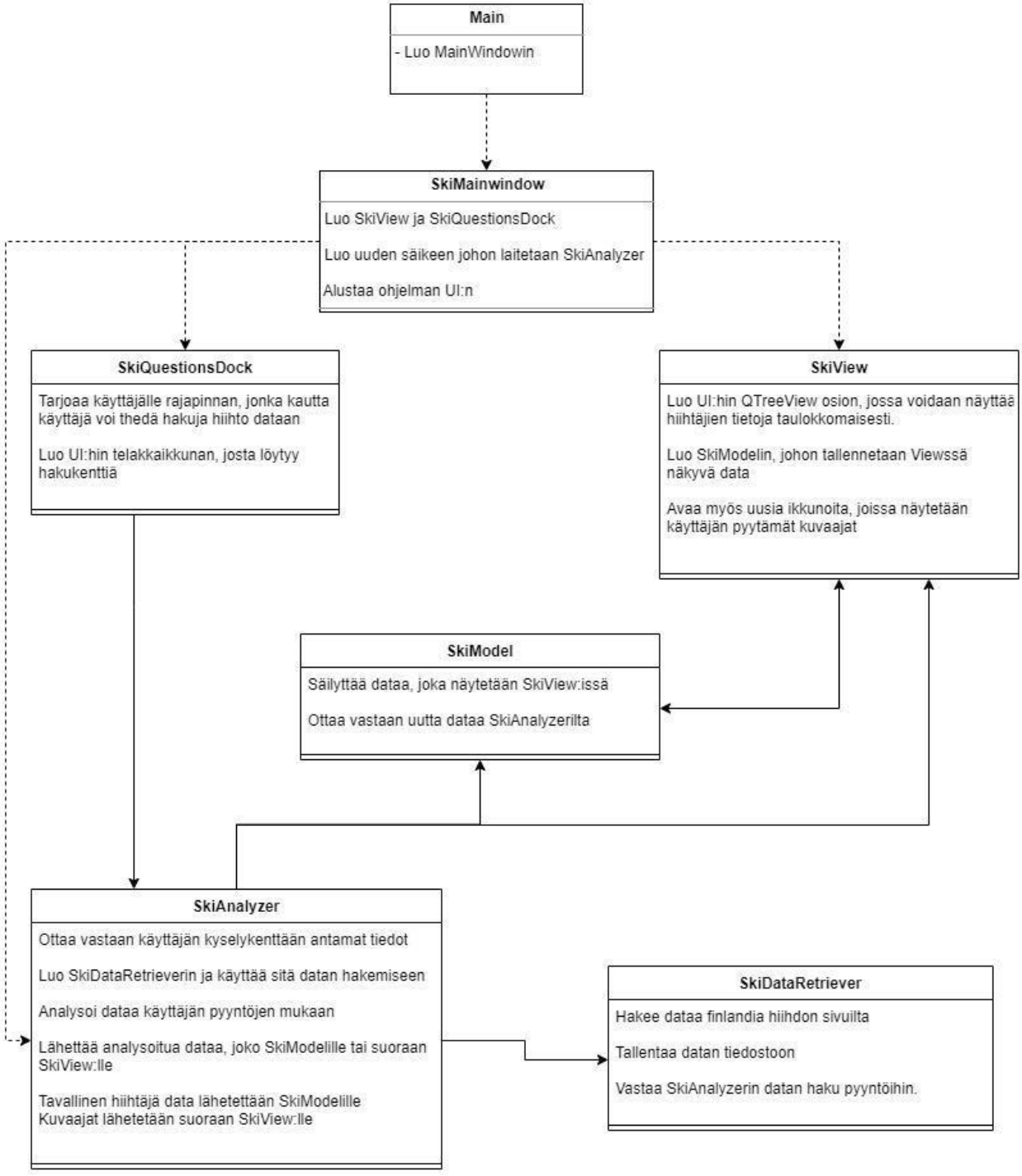
Suunnitteludokumentti: SkiingAnalyzer

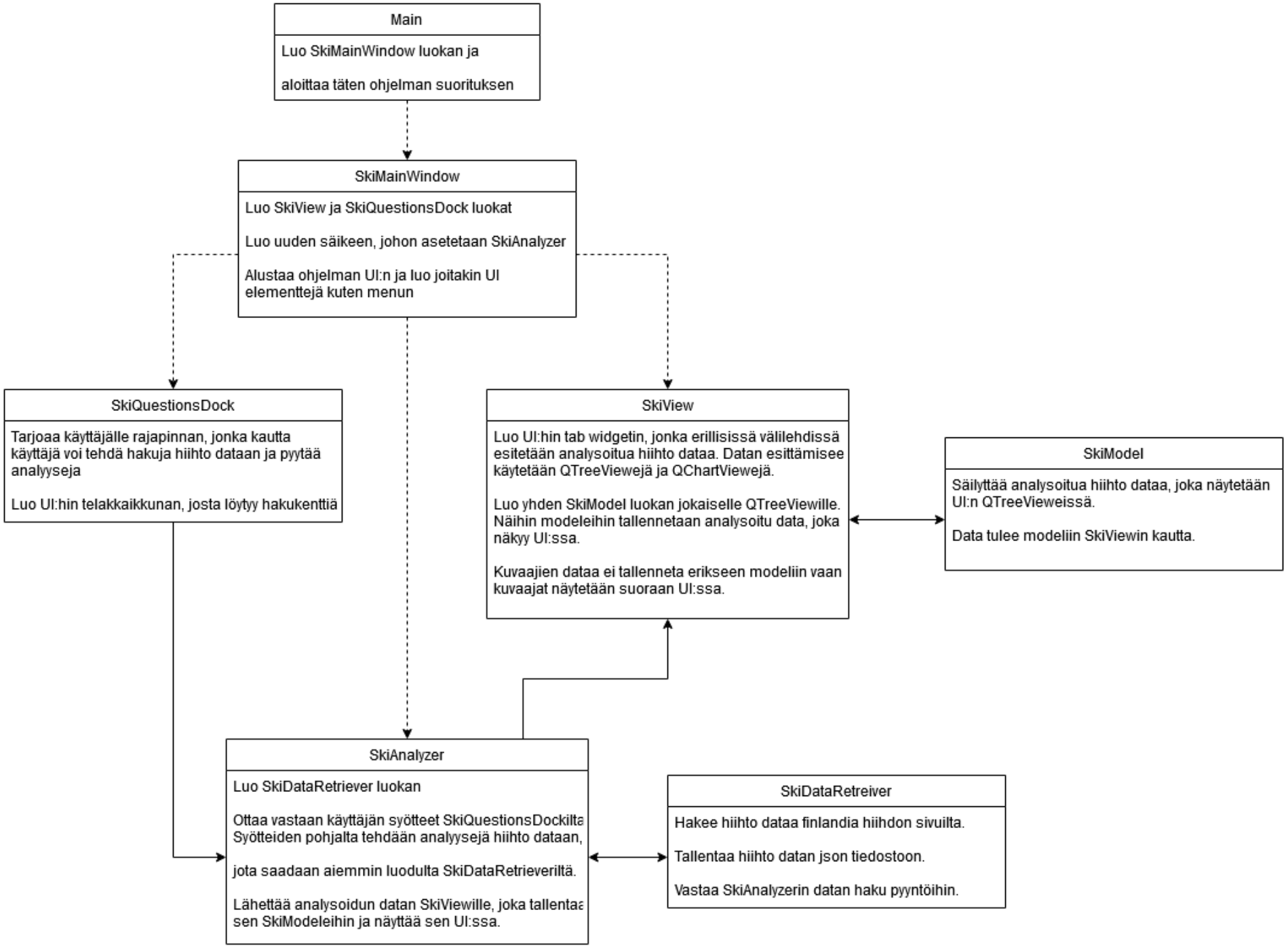
SkiingAnalyzer ohjelma on tarkoitettu Finlandia-hiihdon tulosten analysointiin. Ohjelma koostuu kolmesta erillisestä osasta: käyttöliittymästä, analysointiosuudesta ja rajapinnasta Finlandia-hiihdon tietokantaan. Käyttöliittymän tehtävä on selkeä eli näytetään käyttäjälle dataa ja otetaan vastaan syötteitä. Analysointiosuuden tehtävänä on tarjota käyttöliittymälle analysoitua dataa. Analysointiosuus saa datansa rajapinnan kautta. Rajapinta hakee tietonsa Finlandia-hiihdon tietokannasta ja lähettää tietoa analysoitavaksi tarvittaessa.

Luokat SkiMainwindow, SkiQuestionsDock, SkiView ja SkiModel luovat yhdessä ohjelman käyttöliittymän. SkiMainwindowin tehtävänä on luoda muut käyttöliittymän elementit. SkiQuestionsDockin tehtävänä on ottaa vastaan käyttäjän syötteitä ja lähettää tieto näistä eteenpäin SkiAnalyzerille. SkiViewin tehtävä on luoda SkiModel ja näyttää modeliin tallennettua dataa. SkiModel tallentaa SkiAnalyzerin antaman datan. SkiAnalyzerin tehtävänä on analysoida Finlandia-hiihdon tietokannasta saatua dataa käyttäjän toiveiden mukaisesti. SkiDataRetrieverin tehtävänä on hakea dataa Finlandia-hiihdon tietokannasta ja tarjota tätä dataa SkiAnalyzerille. Ohjelman alkuperäinen rakenne on esitetty kuvassa 1. Kuvassa 2 on puolestaan esitetty ohjelman lopullinen rakenne.

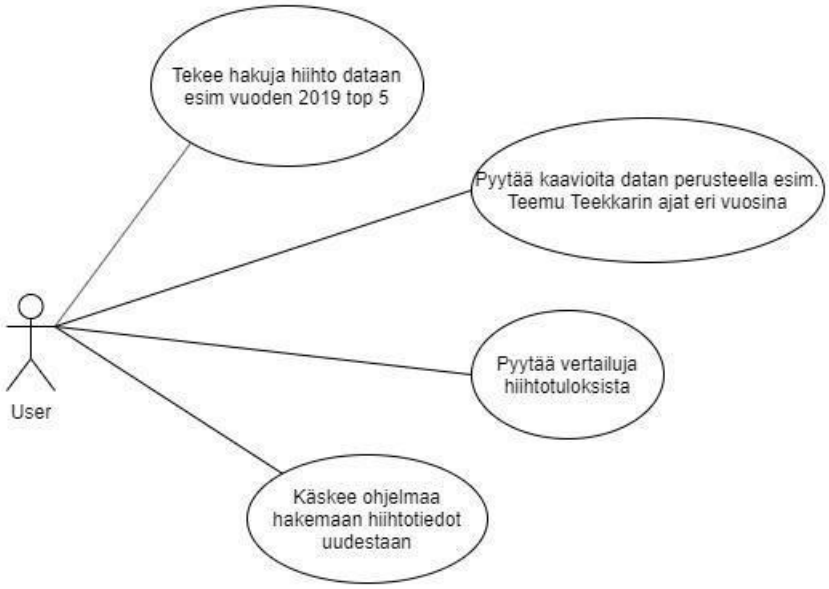
Ohjelmaa varten on laadittu myös käyttötapauskaavio. Kaaviosta näkee mitä käyttäjä voi ohjelmalla tehdä. Käyttökaavio on esitetty kuvassa 3.



Kuva 1: *Kaavio ohjelman rakenteesta.*



Kuva 2: *Uusi kaavio ohjelman rakenteesta.*



Kuva 3: *Ohjelman käyttötapauskaavio*

Rajapintadokumentaatio

Seuraavana on esitetty ohjelman rajapinta Finlandia-hiihdon tietokantaan. Rajapintana siis toimii SkiDataRetriever-luokka, joka hakee tiedot Finlandia-hiihdon tietokannasta ja tallentaa tiedot lokaaliin tiedostoon. Tällöin tiedot on helppo hakea seuraavalla käynnistyskerralla. Tiedot voi päivittää poistamalla lokaalin tiedoston tai käyttöliittymästä löytyvän napin kautta. Rajapinnan funktiot esitetty seuraavassa koodipätkässä.

```
signals:
    /**
     * @brief DataReady: Notifies that the data is retrieved and retriever is
     * ready to serve data
     * @param progress: Number of received and handeled requests
     * @param total: Total number of requests
     */
    void DataReady(int progress, int total);

    /**
     * @brief ParametersReady: Internal signal to notify that post request
     * parameters are valid
     */
    void ParametersReady();

public slots:

    /**
     * @brief GetSkiingData: This method is used to get skiing data from
     *        specific year. Distance specifies the type of the skiing
     *        competition. If distance is empty then the retreiver returns
     *        every result from the specified year.
     * @param year: Year from which the data is fetched
     * @param distance: Specific distance from which the data is fetched
```

```
     * @return Skiing data from a specified year
     * @pre Data is in internal database
     * @post Information found in the database for the year and distance is
     *       returned
     */
    SkiingData GetSkiingData(int year, QString distance = "");

    /**
     * @brief StartSkiingDataRetrieval: Starts the data retrieval
     * @post Data retrieval is started
     */
    void StartSkiingDataRetrieval();

    /**
     * @brief UpdateDataBase: Deletes the local file and starts a new data
     * retrieval and therefore creates the file again with new data.
     * @post Data retrieval is started
     */
    void UpdateDataBase();
```

Sisäiset rajapinnat

Tässä kohdassa on esitettynä ohjelman sisäiset rajapinnat. Ohjelman tärkeimmät rajapinnat ovat SkiQuestionsDockin ja SkiAnalyzerin välinen ja SkiAnalyzerin ja SkiViewin välinen. SkiQuestionsDock lähettää käyttäjän syötteet SkiAnalyzerille, joka analysoi syötteet ja lähettää analysoitua dataa SkiViewille.

SkiAnalyzer

SkiAnalyzer luokan rajapinta:

```
public slots:
    /**
     * @brief run slot starts the new thread that includes this class and the
     *        SkiDataRetriever class.
     */
    void run();

    /**
     * @brief handleSearchRequest searches the database and filters out all
unneeded entries.
     * @param searchParams: the parameters the user has input.
     * @pre parameters are valid, data is accessible.
     * @post the searched data has been emitted.
     */
    void handleSearchRequest(const QVector<QString> &searchParams);

    /**
     * @brief handleCompareRequest seaches database twice and show the result
side-by-side.
     * @param params: user input
     * @pre data is accessible.
     * @post both searches have concluded and data has been emitted.
     */
    void handleCompareRequest(const QVector<QString> &params);
    /**
     * @brief handleTimesRequest searches the results of a single athlete
```

```
     * @param params: the name of the athelete and years to be searched.
     * @pre data is accessible, user has input a name.
     * @post data is emitted.
     */

    void handleTimesRequest(const QVector<QString> &params);

    /**
     * @brief Gives best athlete of chosen gender for each race for the wanted
years.
     * @param params contains: Starting year, end year, gender.
     * @pre   params needs to be QVector<QString> and contain three parameters.
Start year and end year need to be numbers.
     * @post  Emits bestAthleteData. If placement "1" is not found function
emits nothing for that race.
     */
    void handleBestAthleteRequest(const QVector<QString> &params);

    /**
     * @brief Counts all participants by country
     * @param searchyear
     * @pre   param needs to be a number
     * @post  Emits nationalityDistributionData
     */
    void handleCountriesRequest(const QString &param);

    /**
     * @brief Goes through data for given year and race and analyzes the top 10
teams for that race.
     *        Comparison is done by counting the 4 best times of all teams.
     * @param Search year, race.
     * @pre   params has to be a QVector with 2 parameters and params[0] has to
be a number.
     * @post  Emits teamsData.
     */
    void handleTeamsRequest(const QVector<QString> &params);

    /**
     * @brief Gives a prediction of the next years winner by checking past
winners of given race.
     * @param Race.
     * @pre   param data is valid.
     * @post  Emits predictionData.
     */
    void handlePredictionRequest(const QString &param);

signals:

    /**
     * @brief addNewRow signal sends search result data to SkiView
     * @param row: the row of new data to be shown.
     */
    void addNewRow(QVector<QString> row);

    /**
     * @brief compareData sends compare result data to SkiView.
     * @param row: the row of new data to be shown.
     * @param view: the index of the view that should show the data.
     */
    void compareData(QVector<QString> row, int view);
```

```
/**
 * @brief compareNumberOfParticipants signal sends the number of
 *        participants to SkiView.
 * @param numbers: the participant amount that should be shown.
 */
void compareNumberOfParticipants(QPair<QString, QString> numbers);

/**
 * @brief timesData signal sends time progression of a single athlete to
 *        SkiView
 * @param row: the row of new data to be shown.
 */
void timesData(QVector<QString> row);

/**
 * @brief addNewRow signal sends the best athlete's data to SkiView.
 * @param row: the row of new data to be shown.
 */
void bestAthleteData(QVector<QString> row);

/**
 * @brief nationalityDistributionData sends parameters for nationality
 *        distribution graph to SkiView.
 * @param row: the row of new data to be shown.
 */
void nationalityDistributionData(QHash<QString, int> row);

/**
 * @brief teamsData signal sends top ten team data to SkiView.
 * @param row: the row of new data to be shown.
 */
void teamsData(QVector<QString> row);

/**
 * @brief predictionData signal sends the calculated prediction data to
 *        SkiView.
 * @param row: the row of new data to be shown.
 */
void predictionData(QVector<QString> row);

/**
 * @brief dataSent signal informs other classes that the data analysis is
 *        ready.
 * @param index of the tab which contains the new data.
 */
void dataSent(int index);

/**
 * @brief refreshDataStorages signal tells the data retriever to update its
 *        data.
 */
void refreshDataStorages();

/**
 * @brief dataReady informs the main window that the SkiDataRetriever has
 *        completed some of the data retrieval.
 * @param progress: the amount of progress made.
 * @param total: the total amount the progress param can get.
 */
```

```
    void dataReady(int progress, int total);
```

SkiQuestionDock

SkiQuestionsDock-luokan rajapinta:

```
public slots:

    /**
     * @brief releaseButtons slot enables all buttons in the tab indicated by
     *        the param index.
     * @param index: the index of the tab to enable.
     * @post buttons are enabled.
     */
    void releaseButtons(int index);

    /**
     * @brief changeTab slot changes the current tab to the one indicated by
     *        param index.
     * @param index: the index of the tab to change to.
     * @pre  parameter is valid tab index.
     * @post tab has changed to desired one.
     */
    void changeTab(int index);

    /**
     * @brief lockForUpdate slot disables the dock widget.
     * @post dock widget is disabled.
     */
    void lockForUpdate();

    /**
     * @brief releaseAfterUpdate slot enables the dock widget.
     * @post dock widget is enabled.
     */
    void releaseAfterUpdate();

    /**
     * @brief getTimesName slot gets the athlete's name used in the time
     *        development chart from theUI and saves it to the referenced
     *        parameter name.
     * @param name: reference to QString where the name is saved.
     */
    void getTimesName(QString& name);

    /**
     * @brief getDistYear slot gets the year used in the nationality
     *        distribution chart from the UI and saves it to the referenced
     *        parameter name.
     * @param year: reference to QString where the year is saved.
     */
    void getDistYear(QString& year);

signals:
    /**
     * @brief clearView signal clears search view from SkiView.
     */
    void clearView();

    /**
```

```
 * @brief search signal sends search parameters to SkiAnalyzer.
 * @param params includes the needed parameters given by the user.
 */
void search(const QVector<QString> &params);

/**
 * @brief compare signal sends compare parameters to SkiAnalyzer.
 * @param params includes the needed parameters given by the user.
 */
void compare(const QVector<QString> &params);

/**
 * @brief clearCompare signal clears compare view from SkiView.
 */
void clearCompare();

/**
 * @brief getTimes signal sends time progression parameters to SkiAnalyzer.
 * @param params includes the needed parameters given by the user.
 */
void getTimes(const QVector<QString> &params);

/**
 * @brief getBest signal sends best athlete parameters to SkiAnalyzer.
 * @param params includes the needed parameters given by the user.
 */
void getBest(const QVector<QString> &params);

/**
 * @brief clearBest signal clears best athlete view from SkiView.
 */
void clearBest();

/**
 * @brief distribution signal sends nationality distribution parameters to
 *        SkiAnalyzer.
 * @param param includes the needed parameters given by the user.
 */
void distribution(const QString &param);

/**
 * @brief getTeams signal sends team search parameters to SkiAnalyzer.
 * @param params includes the needed parameters given by the user.
 */
void getTeams(const QVector<QString> &params);

/**
 * @brief getPrediction signal sends prediction calculation data to
 *        SkiAnalyzer.
 * @param param includes the needed parameters given by the user.
 */
void getPrediction(const QString &param);

/**
 * @brief clearTeams signal clears teams view from SkiView.
 */
void clearTeams();

/**
 * @brief tabChanged signal informs SkiView about tab change.
```

```
     * @param index: index of the new tab.
     */
    void tabChanged(int index);
```

SkiView

SkiView-luokan rajapinta:

```
public slots:

    /**
     * @brief ClearView slot clears all data from the search tab.
     * @post search view has been cleared.
     */
    void ClearView();

    /**
     * @brief clearCompare slot clears all data from the compare tab.
     * @post compare views have been cleared.
     */
    void clearCompare();

    /**
     * @brief clearTimes slot clears all data from the times tab.
     * @post times view has been cleared.
     */
    void clearTimes();

    /**
     * @brief clearBestAthlete slot clears all data from the best tab.
     * @post best view has been cleared.
     */
    void clearBestAthlete();

    /**
     * @brief clearDistribution slot clears all data from the countries tab.
     * @post distribution view has been cleared.
     */
    void clearDistribution();

    /**
     * @brief clearTeams slot clears all data from the teams tab.
     * @post teams view has been cleared.
     */
    void clearTeams();

    /**
     * @brief Addrow slot receives data from SkiAnalyzer and shows it in the
     *        view.
     * @param row: the row of data to be added to the view.
     * @post model has been notified about new data.
     */
    void Addrow(QVector<QString> row);

    /**
     * @brief showCompareData slot shows compare data.
     * @param row: the row of data to be added to the view.
     * @param model tells which model should take the row's data
     * @pre  param model has to be 1 or 2.
     * @post the right model has been notified about new data.
```

```
 */
void showCompareData(QVector<QString> row, int model);

/**
 * @brief showCompareNumberOfParticipants slot shows the number of
 *        participants above the compare view.
 * @param numbers: includes the numbers that should be shown in the UI.
 * @pre  params are valid.
 * @post params are displayed in the UI.
 */
void showCompareNumberOfParticipants(QPair<QString, QString> numbers);

/**
 * @brief showTimesData slot shows time progression of a single athlete.
 * @param row: the row of data to be added to the view.
 * @pre  param data is valid.
 * @post a chart is shown in the UI.
 */
void showTimesData(QVector<QString> row);

/**
 * @brief showBestAthleteData slot shows the best athlete's data.
 * @param row: the row of data to be added to the view.
 * @post model has been notified about new data.
 */
void showBestAthleteData(QVector<QString> row);

/**
 * @brief showNationalityDistributionData slot shows nationality
 *        distribution graph.
 * @param row: the row of data to be added to the view.
 * @pre  param data is valid.
 * @post a chart is shown in the UI.
 */
void showNationalityDistributionData(QHash<QString, int> row);

/**
 * @brief showTeamsData slot shows top ten teams.
 * @param row: the row of data to be added to the view.
 * @post model has been notified about new data.
 */
void showTeamsData(QVector<QString> row);

/**
 * @brief showPredictionData slot shows the predicted winner.
 * @param row: the row of data to be added to the view.
 * @pre  param is valid.
 * @post data is shown in the UI.
 */
void showPredictionData(QVector<QString> row);

/**
 * @brief dataReady slot resizes columns to contents.
 * @param index of the tab that has new and ready data.
 */
void dataReady(int index);

/**
 * @brief tabChange slot changes tab to the one indicated by the parameter
 *        index.
```

```
     * @param index of the tab to change to.
     * @pre  param is vlid tab index.
     * @post tab has changed to the desired one.
     */
    void tabChange(int index);


    /**
     * @brief saveTimeChart slot saves the current time development chart as
     *        png file. If there is no chart to save then this slot does
     *        nothing.
     */
    void saveTimeChart();

    /**
     * @brief saveDistChart slot saves the current nationality distribution
     *        chart as png file. If there is no chart to save then this slot
     *        does nothing.
     */
    void saveDistChart();

signals:
    /**
     * @brief AddNewRow signal sends new row data to search tab's SkiModel.
     * @param row: the row of data to be added to the model.
     */
    void AddNewRow(QVector<QString> row);

    /**
     * @brief compare1AddRow signal sends new row data to compares tab's first
     *        SkiModel.
     * @param row: the row of data to be added to the model.
     */
    void compare1AddRow(QVector<QString> row);

    /**
     * @brief compare2AddRow signal sends new row data to compares tab's
     *        second SkiModel.
     * @param row: the row of data to be added to the model.
     */
    void compare2AddRow(QVector<QString> row);

    /**
     * @brief bestAddRow signal sends new row data to best tab's SkiModel.
     * @param row: the row of data to be added to the model.
     */
    void bestAddRow(QVector<QString> row);

    /**
     * @brief teamsAddRow signal sends new row data to teams tab's SkiModel.
     * @param row: the row of data to be added to the model.
     */
    void teamsAddRow(QVector<QString> row);

    /**
     * @brief tabHasChanged signal informs SkiQuestionsDock about tab change.
     * @param index of the new tab.
     */
    void tabHasChanged(int index);
```

```
/**
 * @brief lockDockWidget signal orders the dock widget to disable itself
 *        for the duration of the update.
 */
void lockDockWidget();

/**
 * @brief releaseDockWidget signal tells the dock widget to enable itself
 *        after an update has completed.
 */
void releaseDockWidget();


/**
 * @brief getTimesName signal orders the SkiQuestionsDock to return the
 *        name of the athlete used to create the time development chart.
 * @param name: a reference to QString which should get the athlete's name.
 */
void getTimesName(QString& name);

/**
 * @brief getDistYear signal orders the SkiQuestionsDock to return the
 *        year used to create the nationality distribution chart.
 * @param year: a reference to QString which should get the year.
 */
void getDistYear(QString& year);
```

Itsearviointi (välipalautus)

Tekemämme suunnitelma on todettu melko hyväksi ja se on tukenut järjestelmän toteuttamista. Alkuperäinen ”iso kuva” ohjelman toiminnasta oli hyvä ja on pysynyt samana. Todennäköisesti sama suunnitelma tulee pitämään myös toteutuksen loppuun asti. Lähes kaikki toiminnallisuus on tähän asti toteutettu alkuperäisen suunnitelman mukaan. Ainoa muutos tuli UI:hin. Alkuperäisessä suunnitelmassa oli tarkoitus avata UI:ssa uusia ikkunoita, mutta toteutusta tehtäessä päädyttiin vaihtamaan nämä uudet ikkunat SkiViewin välilehdiksi. Ajatus oli, että ohjelma on käyttäjälle selvempi, jos ohjelmassa olisi vain yksi ikkuna ja tämän takia päädyttiin käyttämään QT:n QTabWidgettiä.

Perusteluja suunnitteluratkaisuille (välipalautus)

Ohjelman toteutuksen koodikieleksi valittiin C++ QT ympäristössä, koska kaikki ryhmän jäsenet tunsivat nämä teknologiat jo valmiiksi. UI:ssa päädyttiin hyödyntämään QT:n model view arkkitehtuurin tyyppistä ratkaisua, koska ryhmässä oli yksi jäsen, joka tunsi tämän tekniikan jo entuudestaan. Sama päti QDockWidgetin käyttöön. Ohjelman selkeyden vuoksi päädyttiin ratkaisuun, jossa yksi analysointi luokka vastaan ottaa käyttäjän syötteet UI:lta ja hakee ja analysoi dataa. Tämä nähtiin selkeäksi kokonaisuudeksi ja samalla nähtiin järkeväksi toteuttaa datan haku erillisessä rajapintaluokassa, jotta haluttua tietokantaa olisi helppo vaihtaa.

Itsearviointi loppupalautukseen

Noudatimme suunnitelmaa kaiken muun paitsi UI:n osalta ja siihen tulleet muutokset tehtiin, jo välipalautukseenkin kuvailluista syistä, koska totesimme useamman välilehden olevan käyttäjäystävällisempi ratkaisu kuin useampi ikkuna. Olisimme voineet työn alussa käyttää enemmän aikaa ja ajattelua siihen, että mietimme tarkemmin vielä millä tapaa data on järkevintä näyttää käyttäjälle. Samoin myös analyzer-osion toteutuksen suunnitteluun olisi kannattanut käyttää enemmän aikaa, jolloin olisi ollut mahdollista luoda selkeämpi funktiojako ja kokonaisuus. Kuitenkin isomman linjan suunnitelmamme toimi hyvin ja työn toteuttaminen sujui melko mutkattomasti. Alkuperäinen suunnitelma on tukenut hyvin työn tekemistä ja olemme edenneet suunnitelman mukaisesti. Kaikki alkuperäisessä suunnitelmassa olleet toiminnallisuudet saatiin toteutettua ja lopulta jopa päädyimme toteuttamaan hieman ylimääräisiä ominaisuuksiakin.

Suunnitteluratkaisut

Päädyimme toteuttamaan oman ratkaisumme MVC-arkkitehtuurin ja Qt:n model/view-mallin (https://doc.qt.io/qt-5/model-view-programming.html) pohjalta. Kumpikaan ei suoraan istunut meidän suunnitelmaamme, joten siksi päädyimme omaan malliimme.

SOLID-periaatteet näkyvät toteutuksessamme. Jokaisella luokalla on vain yksi vastuualue ja tähän ratkaisuun päädyimme ohjelman selkeyden ja helpomman muokattavuuden vuoksi. Ohjelmaan on myös helppo toteuttaa lisäominaisuuksia, sillä ohjelma toimii sloteilla ja signaaleilla. Uudet ominaisuudet voidaan siis melko vaivattomasti tuoda ohjelmaan mukaan. Periyttämistä ohjelmassa ei juurikaan käytetä. Periytämme ainoastaan Qt:n omia luokkia.

Lopullinen työnjako

Työnjako meni koko projektin ajalta kuten alussa oli sovittu. Samulin oli tarkoitus hoitaa suurin osa UI:n elementeistä. Eetun oli tarkoitus tehdä ulkoinen rajapinta finlandia hiihtoon. Ville & Villen oli tarkoitus hoitaa datan analysointi ja kuvaajat. Tämän työnjaon voidaan nähdä toteutuneen kokonaisuudessaan.