```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Nihulon2.Model;
using Nihulon2.Model.DbAccess;
using OfficeOpenXml;
using OfficeOpenXml.Style;

namespace Nihulon2.SupervisorsAdministration
{
    /*
     * The class that assumes the layer between the view and DataBase.
     * It provides methods that get data from the dbConnector and
     * calls the view methods to fill its controls with the new data.
     * This controller works with the exams
     */
    public class SupervisorsAdministration_Controller
    {
        private ISupervisorsAdministrationView _view; // instance of the view for data visualization
        private DbConnector dbConn; // An instance of the class that provides connection to the DB

        // The list of exams that are binded to the exam table
        private SortableBindingList<Exam> examsList;
        // The list with exams that have time overlaps
        private SortableBindingList<Exam> examsWithOverlap;

        #region Properties

        /*
         * Filter properties that define what data we need to get from the DB
         */

        // When we need to show exams for a specific period of time,
        // this property is a date of the start of that period
        public string dateFromFilter
        {
            get;
            set;
        }
        // When we need to show exams for a specific period of time,
        // this property is a date of the start of that period
        public string dateToFilter
        {
            get;
            set;
        }

        // If true, show only the exams that refer to a certain division
```

```
        public string divisionFilter
        {
            get;
            set;
        }
        // If true, show only the exams that refer to a certain course
        public string courseFilter
        {
            get;
            set;
        }
        // If true, show only the exams that refer to a certain room
        public string roomFilter
        {
            get;
            set;
        }
        // If true, show the disabled exams
        public bool showDisabledFilter
        {
            get;
            set;
        }
        // If true, show the exams that were added today
        public bool showNewFilter
        {
            get;
            set;
        }

        // Flag specifies in which mode to show exams, All or Overlaps only
        public bool showOverlaps
        {
            get;
            set;
        }

        #endregion

        // The constructor that gets the instance of the view
        // that will be calling to the controller
        public SupervisorsAdministration_Controller (ISupervisorsAdministrationView view)
        {
            // Initializing connection to DB
            dbConn = DbConnector.Instance;
            examsList = new SortableBindingList<Exam>();
            // Binding with the view
            _view = view;
            _view.setController(this);

            this.loadView();

            dbConn.foundOverlaps = false;
        }

```

```
        #region Interface

        // Reload all comboboxes and fill the table
        public void reload()
        {
            reloadFiltersComboBoxes();
            fillTable();
        }

        // Get the exams from the DB and bind the list of exams with the table at the view
        public void fillTable()
        {
            List<Exam> list;

            // Show all exams according to the filters
            if (!showOverlaps)
            {
                // Get exams form DB according to filters
                list = dbConn.getExams(dateFromFilter, dateToFilter, divisionFilter,
                    courseFilter, roomFilter, showDisabledFilter, showNewFilter);
                list.Reverse();
                // Convert the list of exams to the Binding list
                examsList = new SortableBindingList<Exam>(list);
                // Ask the view to bind its table with controller's list of exams
                _view.bindExamsWithTable(ref examsList);
            }
            // Show the exams that have time overlaps
            else
            {
                // Get exams with time overlaps
                list = dbConn.getExamsWithOverlaps();
                // Convert the list of exams to the Binding list
                examsList = new SortableBindingList<Exam>(list);
                _view.bindExamsWithTable(ref examsList);
            }
        }

        // Add a new exam to the DB and reload the table
        public void addExam(Exam newExam)
        {
            // Insert the new exam
            dbConn.insertExam(newExam);
            // Fill the table with exams
            fillTable();
        }

        // Delete an exam from the DB
        public void deleteExam(int examId)
        {
            dbConn.removeExam(examId);
        }

```

```
        // Save changed exam to the DB and reload the table
        public void changeExam(Exam changedExam)
        {
            dbConn.updateExam(changedExam);
            // Check time overlaps if the flag "Show overlaps" is true
            if(this.showOverlaps == true)
                dbConn.markExamsWithOverlap();
            fillTable();
        }

        // Takes a name of Excel file and create the file
        // with exam table
        public void createExcel(string excelName)
        {
            string filePath;
            if(excelName != "")
            {
                // the path to the excel file
                filePath = @"./Excel files/" + excelName + ".xlsx";

                // Creating the directory for Excel files if doesn't exists
                if (!Directory.Exists(@"./Excel files/"))
                    Directory.CreateDirectory(@"./Excel files/");

                // Create an instance of the excelPackage, create the file and fill it with the exams
                using (ExcelPackage excel = new ExcelPackage())
                {
                    // create a workSheet
                    excel.Workbook.Worksheets.Add("Worksheet1");
                    // get the created workSheet
                    ExcelWorksheet excelWorksheet = excel.Workbook.Worksheets["Worksheet1"];

                    // create the row of headers
                    List<string[]> headerRow = new List<string[]>()
                    {
                        new string[] { "תוספת זמן", "שעת סיום", " שעת התייצבות", "חדר", "מקצוע", "קבוצה", "מגמה/קורס", " חטיבה", "שם משגיח/ה", "תאריך בחינה", "מס", "בוטל", " הערה", "זמן" },
                    };
                    // Insert header row data
                    excelWorksheet.Cells["A1"].LoadFromArrays(headerRow);
                    // Build the data for inserting from the exams at the table
                    List<string[]> examsData = this.getExamsForExcel();
                    excelWorksheet.Cells["A2"].LoadFromArrays(examsData);

                    // format the workSheet
                    this.formatExcel(excelWorksheet, examsData.Count);

                    FileInfo excelFile = new FileInfo(filePath);
                    excel.SaveAs(excelFile);
                }
            }
```

```
        }

        // Load exams from Excel file
        // and save the changes into the DB
        public void loadFromExcel(string path)
        {
            // Create an instance of the excelPackage from the Excel file
            using (ExcelPackage excel = new ExcelPackage(new FileInfo(path)))
            {
                int i;
                // Get worksheet
                ExcelWorksheet workSheet = excel.Workbook.Worksheets[1];

                // Read the row of headers from the excel
                char ch = 'A'; // Start from A

                // Fill the array with headers
                string[] headers = new string[14];
                for(i = 1; i <= headers.Length; i++)
                {
                    if (workSheet.Cells[$"{(char)(ch+i-1)}1"].Value != null)
                        headers[i-1] = workSheet.Cells[$"{(char)(ch + i - 1)}1"].Value.ToString();
                    else
                        headers[i-1] = "";
                }

                // Check if the headers at correct order and format
                if (this.checkExcelFormat(headers))
                {
                    // Save the exams from that worksheet to the DB
                    this.loadExamsFromExcelToDB(workSheet);

                }
            }
        }

        // Finds exams with time overlaps
        public void checkOverlaps()
        {
            dbConn.markExamsWithOverlap();
            this.fillTable();
        }

        #endregion


        #region Private methods for internal use
        // Get names of related items from DB and fill the combo boxes of the filters in the view
        private void reloadFiltersComboBoxes()
        {
            string[] divisions, courses, rooms;

            // Get the names of related items from the DB
            divisions = dbConn.getRelatedItemsNamesByType("חטיבות");
```

```
            courses = dbConn.getRelatedItemsNamesByType("מגמות");
            rooms = dbConn.getRelatedItemsNamesByType("חדרים");

            // Clear combo boxes
            _view.clearComboBoxes();

            // Fill the combo boxes with names of related items
            _view.fillDivisionsCbo(divisions);
            _view.fillCoursesCbo(courses);
            _view.fillRoomsCbo(rooms);
        }

        // Set default values for the properties
        private void setDefaultFilters()
        {
            this.dateFromFilter = "";
            this.dateToFilter = "";
            this.divisionFilter = "הכול";
            this.courseFilter = "הכול";
            this.roomFilter = "הכול";
            this.showDisabledFilter = false;
        }

        // Builds rows with exams for inserting into the excel file
        private List<string[]> getExamsForExcel()
        {
            List<string[]> data = new List<string[]>();

            // Go through the exams at the table and fill the rows data
            foreach(Exam exam in examsList)
            {
                string[] row = new string[14];

                row[0] = exam.hasExtraTime ? "יש" : "";
                row[1] = exam.EndingTime;
                row[2] = exam.StartTime;
                row[3] = exam.room;
                row[4] = exam.DisciplineName;
                row[5] = exam.GroupName;
                row[6] = exam.course;
                row[7] = exam.division;
                row[8] = exam.SupervisorName;
                row[9] = exam.Date;
                row[10] = exam.Id.ToString();
                row[11] = exam.isCanceled ? "בוטל" : "";
                row[12] = exam.Comments;

                // Calculate difference between the start time and the ending time
                // for getting the time scale of the exam
                DateTime fromTime = DateTime.Parse(exam.StartTime);
                DateTime toTime = DateTime.Parse(exam.EndingTime);
                TimeSpan hours = (toTime - fromTime);
                row[13] = hours.ToString();

                data.Add(row);
```

```
            }
            return data;
        }

        // Gets workSheet of Excel with exams and formats its cells
        private void formatExcel(ExcelWorksheet excelWorksheet, int numRows)
        {
            int i;

            excelWorksheet.View.RightToLeft = true;

            // Set width for the columns
            excelWorksheet.Column(1).Width = 7;
            excelWorksheet.Column(2).Width = 10;
            excelWorksheet.Column(3).Width = 10;
            excelWorksheet.Column(4).Width = 12;
            excelWorksheet.Column(5).Width = 15;
            excelWorksheet.Column(6).Width = 10;
            excelWorksheet.Column(7).Width = 25;
            excelWorksheet.Column(8).Width = 10;
            excelWorksheet.Column(9).Width = 17;
            excelWorksheet.Column(10).Width = 13;
            excelWorksheet.Column(11).Hidden = true;
            excelWorksheet.Column(12).Width = 7;
            excelWorksheet.Column(13).Width = 25;
            excelWorksheet.Column(14).Width = 10;

            // Set wrap text
            for (i = 1; i <= 14; i++)
                excelWorksheet.Column(i).Style.WrapText = true;

            // Set border style
            excelWorksheet.Cells[$"A1:N{numRows + 1}"].Style.Border.Left.Style = OfficeOpenXml.Style.ExcelBorderStyle.Thin;
            excelWorksheet.Cells[$"A1:N{numRows + 1}"].Style.Border.Right.Style = OfficeOpenXml.Style.ExcelBorderStyle.Thin;
            excelWorksheet.Cells[$"A1:N{numRows + 1}"].Style.Border.Top.Style = OfficeOpenXml.Style.ExcelBorderStyle.Thin;
            excelWorksheet.Cells[$"A1:N{numRows + 1}"].Style.Border.Bottom.Style = OfficeOpenXml.Style.ExcelBorderStyle.Thin;
            excelWorksheet.Cells[$"A1:N{numRows + 1}"].Style.Border.BorderAround (OfficeOpenXml.Style.ExcelBorderStyle.Thick);
            // Paint yellow the empty cells at the column with names of supervisors
            for (i = 2; i <= numRows + 1; i++)
            {
                if(excelWorksheet.Cells[$"I{i}"].Value == null)
                {
                    excelWorksheet.Cells[$"I{i}"].Style.Fill.PatternType = OfficeOpenXml.Style.ExcelFillStyle.Solid;
                    excelWorksheet.Cells[$"I{i}"].Style.Fill.BackgroundColor.SetColor (System.Drawing.Color.Yellow);
```

```
                }
            }
        }

        // Checks format of the excel file
        private bool checkExcelFormat(string[] headers)
        {
            string[] patternHeaders = { "תוספת זמן", "שעת סיום", " שעת התייצבות", "חדר", "מקצוע", "קבוצה",
                "מגמה/קורס", "חטיבה", "שם משגיח/ה", "תאריך בחינה", "מס", "בוטל", "הערה", "זמן" };
            // Check the header row if the columns at correct order
            for(int i = 0; i < 14; i++)
            {
                if (headers[i] != patternHeaders[i])
                    return false;
            }
            return true;
        }

        // Sets default filters and fill the table with exams
        private void loadView()
        {
            // Set default filters
            setDefaultFilters();
            // Get names of related items and fill the combo boxes of the filters in the view
            reloadFiltersComboBoxes();

            // Bind the event handler that catch time overlaps with the processing method of the view
            dbConn.onOverlapsStateChanged += _view.showOverlapsWarning;

            // Fill the table with exams
            fillTable();
        }

        // Takes an Excel worksheet with exams data as a parameter
        // and saves the exams from that worksheet to the DB
        private void loadExamsFromExcelToDB(ExcelWorksheet workSheet)
        {
            // Array for saving id, supervisor name and comments
            string[] examData = new string[3];
            // Go through all rows at the excel file and save the changes into the DB
            for (int i = 2; i <= workSheet.Dimension.End.Row; i++)
            {
                // Get data from the excel
                // Id
                if (workSheet.Cells[$"K{i}"].Value != null)
                    examData[0] = workSheet.Cells[$"K{i}"].Value.ToString();
                else
                    examData[0] = "";
                // Supervisor name
                if (workSheet.Cells[$"I{i}"].Value != null)
                    examData[1] = workSheet.Cells[$"I{i}"].Value.ToString();
```

```
                else
                    examData[1] = "";
                // Comments
                if (workSheet.Cells[$"M{i}"].Value != null)
                    examData[2] = workSheet.Cells[$"M{i}"].Value.ToString();
                else
                    examData[2] = "";

                // Save the changes into the DB
                dbConn.updateExamFromExcel(examData);
            }
        }

        #endregion

    }
}

```