**Примерно индивидуално практическо задание № 1**

Държавният изпит – част по практика на професията и специалността, се състои в създаване на софтуерен продукт, който задължително включва следните три компонента:

1. потребителски интерфейс;

2. система за управление на бази от данни;

3. софтуерна система, базирана на трислойния модел, която координира потребителския интерфейс и системата за управление на бази от данни.

**1. Примерно индивидуално практическо задание № 1: Създайте софтуерен продукт за управление на записването на часове в зъболекарска клиника.**

Продуктът трябва да поддържа информация за работещите зъболекари в клиниката, като за всеки от тях се съхранява информация:

● име

● фамилия

● описание

● списък с регистрирани пациенти

За всеки пациент се съхранява информация:

● име

● фамилия

● възраст

● дата на раждане

● списък от извършените прегледи

● основен зъболекар

За всеки преглед се записва информация:

● пациент, на когото се извършва прегледа

● дата и час на извършване на прегледа

● зъболекар, който е извършил прегледа

● описание в свободен текст

Продуктът трябва да разполага с модул, който запазва час за зъболекар. За резервиране на час е нужна следната информация:

● пациент

● дата и час

● очаквана продължителност на прегледа

● зъболекар.

Важно: Не трябва да се допуска двама различни пациенти да имат еднакви или частично припокриващи се часове при един и същ зъболекар!

Продуктът трябва да разполага и с основен статистически модул, който показва следната информация:

● брой работещи зъболекари

● брой регистрирани пациенти в клиниката

● брой извършени прегледи

● брой предстоящи записани часове

Общи изисквания:

● Продуктът трябва да разполага с подходящ интерфейс за управление на

информацията в него.

● Данните трябва да се съхраняват в база данни.

● Архитектурата на приложението трябва да следва принципите на трислойната архитектура или да ги надгражда.

● Продуктът трябва да бъде създаден, чрез спазване на добрите практики за софтуерна разработка и писане на код.

**2. Указания (инструкции/изисквания) за изпълнение на практическото задание:**

Ученикът/обучаваният трябва да:

- избере и използва подходяща софтуерна среда за разработка спрямо изискванията на индивидуалното практическо задание и спрямо изучаваните софтуерни технологии;

- спазва правилна структура на софтуерния проект;

- спазва добрите практики при писане на програмен код;

- избере подходяща архитектура за реализация на решението;

- създаде подходящ модел на база данни, който да е съобразен с връзките между отделните единици в системата;

- създаде подходящ и удобен потребителски интерфейс за работа с продукта, спазвайки добрите практики за разработка на потребителски интерфейс;

- демонстрира коректната работа на проекта.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Обяснителни бележки и указания за решаване на задачата.***

Обяснителни бележки и указания за решаване на задачата Индивидуално задание 1.

Необходими са инсталирани MySQL база данни и MS Visual Studio.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Създаване на БД на MySQL**

Първо ще започнем със създаването на базата от данни.

От условието на заданието се вижда, че имаме нужда от 3 таблици – за зъболекари, за пациенти и за часове за прегледи. Условието, така както е зададено, не е точно и ясно специфицирано и съдържа неясноти и неточности. Те трябва да се изчистят преди да създадем базата.

В таблицата за зъболекари можем да имаме полета за име, фамилия и описание, но не може да вкараме списък с регистрирани пациенти в едно поле – това е отношение едно към много. Това ще стане с референтно поле в таблицата за пациентите, което реферира към зъболекар – един пациент има един основен зъболекар, но един зъболекар може да има няколко пациента.

Подобно е положението и в таблицата за пациентите – не може да имаме списък с извършените прегледи. В таблицата за прегледите ще имаме полета, които реферират към пациента и зъболекаря съответно. Ако се нуждаем от дадения списък, той може да се извлече със SELECT и задаване на подходящо условие.

Друг проблем е излишък – възраст и дата на раждане за пациент. Това е ненужно – може да се съхранява само датата на раждане – възрастта се изчислява.

В таблицата за прегледите се споменава за дата и час за прегледа (начало на прегледа), но не се споменава, че той има и продължителност. По-нататък в условието, обаче, се споменава, че при резервиране на час, трябва да се въвежда и очаквана продължителност на прегледа. Ние трябва да съхраняваме това в базата или като продължителност, или като очакван край на прегледа.

Всичко това води да реорганизиране на структурата на базата. Крайният вариант е показан по-долу. Не трябва да забравяме да добавим и полета id, които да служат за първичен ключ.

С MySQL Workbench отворете конекция и създайте нова БД.

Изпълнете командите:

CREATE DATABASE Dentist\_Clinic\_DB;

USE Dentist\_Clinic\_DB;

CREATE TABLE dentists

(

id INT NOT NULL AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

family VARCHAR(50) NOT NULL,

description VARCHAR(1000),

CONSTRAINT pk\_dentists PRIMARY KEY (id)

);

CREATE TABLE patients

(

id INT NOT NULL AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

family VARCHAR(50) NOT NULL,

birthdate DATE NOT NULL, -- age can be calculated

dentist\_id INT NOT NULL, -- ref to main dentist

CONSTRAINT pk\_patients PRIMARY KEY (id),

CONSTRAINT fk\_patients\_dentists FOREIGN KEY (dentist\_id) REFERENCES dentists (id)

);

CREATE TABLE examinations

(

id INT NOT NULL AUTO\_INCREMENT,

patient\_id INT NOT NULL, -- ref to patient

dentist\_id INT NOT NULL, -- ref to dentist

appointment DATETIME NOT NULL, -- appointment / reservation

exam\_end DATETIME NOT NULL, -- appointment + interval (expected duration)

notes VARCHAR(2000),

CONSTRAINT pk\_examinations PRIMARY KEY (id),

CONSTRAINT fk\_examinations\_patients FOREIGN KEY (patient\_id) REFERENCES patients (id),

CONSTRAINT fk\_examinations\_dentists FOREIGN KEY (dentist\_id) REFERENCES dentists (id)

);

Вкарайте адекватни примерни данни в таблиците.

Пример:

INSERT INTO dentists(name, family)

VALUES

('Roberto', 'Jovanni'),

('Ivan', 'Petrov'),

('Yana', 'Ivanova');

INSERT INTO patients(name, family, birthdate, dentist\_id )

VALUES

('John', 'Smith', '1971-05-23', 1),

('Petar', 'Petrov', '1972-12-12', 3),

('Ana', 'Antonova', '1982-02-21', 2),

('Boris', 'Borisov', '1977-09-11', 2),

('Georgi', 'Georgiev', '2001-06-22', 1),

('Svetla', 'Svetlinova', '1991-04-18', 3),

('Viktor', 'Krumov', '1992-05-05', 2);

INSERT INTO examinations(patient\_id , dentist\_id , appointment , exam\_end )

VALUES

(1, 1, '2021-08-14 10:30:00', '2021-08-14 10:55:00'),

(3, 2, '2021-08-14 11:00:00', '2021-08-14 11:25:00'),

(4, 2, '2021-08-14 11:30:00', '2021-08-14 11:55:00'),

(6, 3, '2021-08-15 09:30:00', '2021-08-14 09:55:00'),

(1, 1, '2022-08-14 10:30:00', '2022-08-14 10:55:00');

Проверете дали данните са правилно вкарани.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Създаване на приложение с MS Visual Studio на C#**

Приложението трябва да следва трислойния модел - слой за данни, слой за бизнес логика и слой за потребителски интерфейс.

Базата данни ще бъде на локалната машина (localhost - 'server=127.0.0.1;uid=root;pwd=1234;').

За потребителски интерфейс ще използваме WinForms.

Отворете Visual Studio и създайте нов проект - Create New Project.

За език изберете C#. Платформа - Windows.

За тип проект изберете Windows Forms Application (.NET Framework). Цъкнете Next.

Напишете името на проекта и солюшъна и къде да се запази.

Пример: DentalClinicForm за име на проект и DentalClinicApp1 за солюшън.

Изберете .NET Framework 4.7.2

Цъкнете Create.

Изчакайте новият проект да се създаде.

VS автоматично създава празна нова форма наречена Form1. Това ще бъде графичния потребителски интерфейс. Засега ще го оставим така.

Създайте 2 нови проекта - за данни и бизнес логика.

От Solution Explorer цъкнете солюшъна с десен бутон и изберете Add -> New Project.

За тип проект изберете Class Library (.NET Framework). Променете името на проекта от ClassLibrary1 на DataLib.

Направете същото и създайте проект BusinessLib.

Освен това добавете следните референции (десен бутон върху проекта -> Add -> Reference) за всеки от проектите:

На проект BusinessLib добавете референция към DataLib.

На проект DentalClinicForm добавете референция към BusinessLib и DataLib.

\_\_\_\_\_\_\_\_\_\_\_\_\_

**Слой за данни**

В проекта DataLib също трябва да се добави референция към конектор за MySQL.Data

Изберете Browse и го намерете:

C:\Program Files (x86)\MySQL\Connector NET 8.0\Assemblies\v4.5.2\MySql.Data.dll

Изберете го и цъкнете Add и OK.

В проекта DataLib променете името на Class1.cs на DataClass.cs.

(десен бутон върху Class1.cs -> Rename)

Ще попита дали да се смени името в кода -> Yes.

В кода на файла DataClass.cs добавете:

using MySql.Data.MySqlClient;

В класа class DataClass добавете:

-----------------

public class DataClass

{

private MySqlConnection dbConn;

private string commandString;

private MySqlCommand mySqlCmd;

private MySqlDataReader reader;

public DataClass()

{

dbConn = new MySqlConnection("server=127.0.0.1;uid=root;pwd=1234;" +

"database=Dentist\_Clinic\_DB;");

}

public void Command(string cmd)

{

commandString = cmd;

dbConn.Open();

using (dbConn)

{

mySqlCmd = new MySqlCommand(commandString, dbConn);

}

}

public object GetScalar()

{

dbConn.Open();

using (dbConn)

{

return mySqlCmd.ExecuteScalar();

}

}

public List<object> GetQuery(List<string> fieldNames)

{ // Gets query result - returns list of things for 1 column

// Returns list of list of things for multiple columns

List<object> list = new List<object>();

dbConn.Open();

using (dbConn)

{

reader = mySqlCmd.ExecuteReader();

using (reader)

{

while( reader.Read() )

{

if(fieldNames.Count == 1)

{

list.Add(reader[fieldNames[0]]);

}

else

{

List<object> subList = new List<object>();

foreach (string fieldName in fieldNames)

{

subList.Add(reader[fieldName]);

}

list.Add(subList);

}

}

}

}

return list;

}

public int ExecuteCommand()

{ // execute non query command - INSERT, UPDATE...

// returns number of affected rows

dbConn.Open();

using (dbConn)

{

return mySqlCmd.ExecuteNonQuery();

}

}

}

-------------------

Command(string cmd) - Това задава SQL командата, която ще се изпълнява.

GetScalar() - Изпълнява командата и връща скалар - единична стойност.

ExecuteCommand() - Изпълнява команда, която не е заявка - напр. INSERT

GetQuery(List<string> fieldNames) - Изпълнява SELECT команда. В списък fieldNames са имената на полетата от заявката/командата - те трябва да съвпадат и да са в същата последователност.

Ако полето е само едно резултата е списък от полета от тип object - трябва да се конвертират в правилен тип преди да се ползват.

Ако полетата са няколко резултата е списък от подсписъци от тип object. Всеки подсписък съответства на запис/ред, а всеки елемент в него на поле/колона. Трябва да се извършва конверсия от object в подходящ тип.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Слой за бизнес логика**

В проекта BusinessLib променете името на Class1.cs на BusinessClass.cs.

Ще попита дали да се смени името в кода -> Yes.

В кода на файла BusinessClass.cs добавете:

using DataLib;

В класа BusinessClass се имплементира основната логика на приложението.

В класа BusinessClass се добавя инстанция на DataClass, която се инициализира в конструктора:

public class BusinessClass

{

private DataClass dataConn;

public BusinessClass()

{

dataConn = new DataClass();

}

}

В класа BusinessClass се добавят методи имплементиращи изискванията.

Заданието има 2 основни изисквания (подзадачи) - да се прави нова резервация и да се извежда статистическа информация. Ще започнем от второто, защото е по-лесно.

Продуктът трябва да разполага и с основен статистически модул, който показва следната информация:

брой работещи зъболекари

public long GetDentistsCount()

{

dataConn.Command("SELECT COUNT(\*) FROM dentists");

return (long)dataConn.GetScalar();

}

брой регистрирани пациенти в клиниката

public long GetPatientsCount()

{

dataConn.Command("SELECT COUNT(\*) FROM patients");

return (long)dataConn.GetScalar();

}

брой извършени прегледи

public long GetDoneExamsCount()

{

dataConn.Command("SELECT COUNT(\*) FROM examinations\n" +

"WHERE exam\_end < now();");

return (long)dataConn.GetScalar();

}

брой предстоящи записани часове

public long GetFutureExamsCount()

{

dataConn.Command("SELECT COUNT(\*) FROM examinations\n" +

"WHERE appointment > now();");

return (long)dataConn.GetScalar();

}

Методите са еднотипни - първо се формира SQL командата, след това се изпълнява и връща скалар - една стойност.

Не забравяйте да сложите символ за нов ред "\n" в низа на командата, където е необходимо.

В класа BusinessClass се добавя функционалност имплементираща изискванията:

Продуктът трябва да разполага с модул, който запазва час за зъболекар.

Важно: Не трябва да се допуска двама различни пациенти да имат еднакви или частично припокриващи се часове при един и същи зъболекар!

В класа BusinessClass се добавят полета и характеристики съхраняващи данните за пациенти и зъболекари в списъци:

private List<string> patientNames;

private List<int> patientIDs;

private List<int> patient\_dentistIDs;

private List<string> dentistNames;

private List<int> dentistIDs;

public List<string> PatientNames { get { return patientNames; } }

public List<int> PatientIDs { get { return patientIDs; } }

public List<int> PatientDentistIDs { get { return patient\_dentistIDs; } }

public List<string> DentistNames { get { return dentistNames; } }

public List<int> DentistIDs { get { return dentistIDs; } }

В класа BusinessClass се добавят методи за извличане на данните за пациенти и зъболекари:

public void GetAllPatients()

{

List<object> list;

dataConn.Command("SELECT id, CONCAT(name, ' ', family) as fullname, dentist\_id\n " +

"FROM patients ;");

list = dataConn.GetQuery(new List<string>() { "id", "fullname", "dentist\_id" });

patientIDs = new List<int>();

patientNames = new List<string>();

patient\_dentistIDs = new List<int>();

foreach (object item in list)

{

patientIDs.Add((int)((List<object>)item)[0]);

patientNames.Add((string)((List<object>)item)[1]);

patient\_dentistIDs.Add((int)((List<object>)item)[2]);

}

}

public void GetAllDentists()

{

List<object> list;

dataConn.Command("SELECT id, CONCAT(name, ' ', family) as fullname\n " +

"FROM dentists ;");

list = dataConn.GetQuery(new List<string>() { "id", "fullname" });

dentistIDs = new List<int>();

dentistNames = new List<string>();

foreach (object item in list)

{

dentistIDs.Add((int)((List<object>)item)[0]);

dentistNames.Add((string)((List<object>)item)[1]);

}

}

Добавя се метод за проверка на колизии при резервацията на часове:

public long CheckAppointmentForCollisions(int dentID, string start, string end)

{

dataConn.Command("SELECT COUNT(\*) \nFROM examinations\n" +

$"WHERE dentist\_id = {dentID} \n" +

$" AND ((\'{start}\' BETWEEN appointment AND exam\_end)\n" +

$" OR(\'{end}\' BETWEEN appointment AND exam\_end)\n" +

$" OR(appointment BETWEEN \'{start}\' AND \'{end}\')\n" +

$" OR(exam\_end BETWEEN \'{start}\' AND \'{end}\'));\n");

return (long)dataConn.GetScalar();

}

Добавя се метод за вкарване на нова резервация за час за преглед:

public int AddAppointment(int patientID, int dentistID, string appointmentStart, string appointmentEnd)

{

dataConn.Command("INSERT INTO examinations(patient\_id , dentist\_id , appointment , exam\_end )\n " +

"VALUES\n " +

$"({patientID}, {dentistID}, '{appointmentStart}', '{appointmentEnd}');");

return dataConn.ExecuteCommand();

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Презентационнен слой**

От Solution Explorer избираме DentalClinicForm -> Form1.cs

В централния прозорец се показва формата Form1.cs[Design]

От менюто View -> Toolbox се показват инструментите (визуални компоненти).

Избираме бутон и го поставяме върху формата.

Бутонът по подразбиране се казва button1. От панела Properties намираме Name и променяме името на бутона на btnStat. По същия начин полето Text = 'Basic Statistics'. Текста върху бутона ще се промени. Може да промените размера и местоположението на бутана.

От панела Properties избираме Events (икона - светкавица) и цъкаме двукратно върху събитие Click.

Автоматично се генерира празен метод btnStat\_Click и се асоциира със събитието за кликване на бутона btnStat. Ние трябва да добавим код в метода.

Във Form1.cs се добавя using BusinessLib;

Във класа Form1 се добавя поле за инстанция на BusinessClass :

private BusinessClass business = new BusinessClass();

В празния метод btnStat\_Click добавяме код, който извлича и показва статистиката:

private void btnStat\_Click(object sender, EventArgs e)

{

string caption = "Basic Statistics";

string message = "";

message += "Working dentists count = " + business.GetDentistsCount() + "\n";

message += "Registered patients count = " + business.GetPatientsCount() + "\n";

message += "Exams done = " + business.GetDoneExamsCount() + "\n";

message += "Future appointments = " + business.GetFutureExamsCount() + "\n";

MessageBoxButtons btns = MessageBoxButtons.OK;

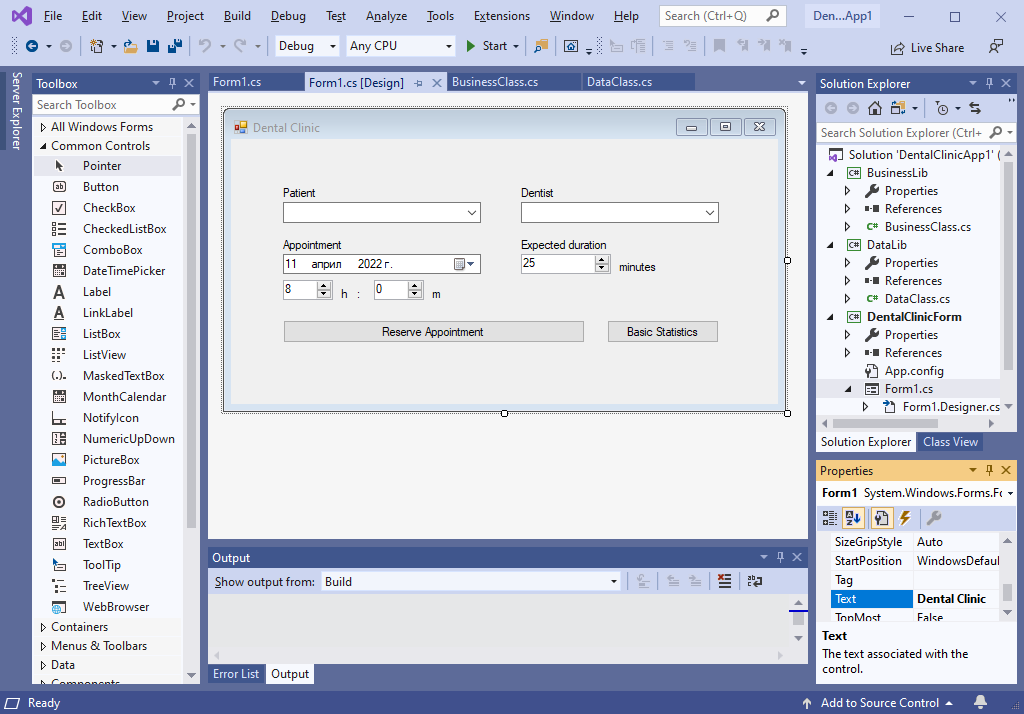
MessageBox.Show(message, caption, btns);

}

С това статистическото подусловие на задачата е завършено.

Трябва да се направи другото подусловие - за добавяне на нова резервация.

В дизайна на формата Form1.cs[Design] добавете визуални контролни компоненти от Toolbox. (Виж картинка Form1.png)



Добавете компоненти - 2 ComboBox, 1 DateTimePicker, 3 NumericUpDown, 1 Button и няколко (7) етикета Label.

Подредете компонентите и променете техните имена и характеристики от Properties.

ComboBox comboPatient

ComboBox comboDentist

Label lblPatient

Label lblDentist

DateTimePicker dateAppointmentPicker

Label lblAppointment

Label lblDuration

NumericUpDown numDurationMinutes

Label lblMinutes

NumericUpDown numHour

NumericUpDown numMinutes

Button btnReserveAppointment

Бележка: по подразбиране компонентите се именоват автоматично с името на класа с малка буква и добавен номер (comboBox1, comboBox2, ...) - имената се променят за да бъдат по смислени (comboPatient, comboDentist, ...).

За бутона btnReserveAppointment променете полето Text = 'Reserve Appointment'.

За numHour променете Minimum = 8, Maximum = 16.

За numMinutes променете Increment = 5, Maximum = 55.

За numDurationMinutes променете Increment = 5, Value = 25.

Селектирайте цялата форма Form1 и от Properties -> Events цъкнете двукратно върху Load. Автоматично се генерира празен метод Form1\_Load и се асоциира със събитие за зареждане на формата. Добавете код:

private void Form1\_Load(object sender, EventArgs e)

{

business.GetAllPatients();

comboPatient.DataSource = business.PatientNames;

business.GetAllDentists();

comboDentist.DataSource = business.DentistNames;

}

.

Това чете информацията за пациенти и зъболекари и асоциира полето DataSource със списъците от PatientNames и DentistNames съответно. Това ще направи списъците с имена в комбо боксовете да се зареждат автоматично при зареждане на формата.

Селектирайте comboPatient и от Properties -> Events цъкнете двукратно върху SelectedIndexChanged. Автоматично се генерира празен метод comboPatient\_SelectedIndexChanged. Той ще се извиква всеки път когато се промени селектираният елемент от comboPatient. Добавете код:

private void comboPatient\_SelectedIndexChanged(object sender, EventArgs e)

{

int dentID = business.PatientDentistIDs[comboPatient.SelectedIndex];

int dentIdx = business.DentistIDs.IndexOf(dentID);

if (dentIdx != -1)

comboDentist.SelectedIndex = dentIdx;

}

.

Когато се промени селектирания индекс на пациента автоматично се променя селекцията на зъболекаря с тази на личния зъболекар на пациента.

Сега добавете още един ред в края на Form1\_Load, който извиква comboPatient\_SelectedIndexChanged.

private void Form1\_Load(object sender, EventArgs e)

{

business.GetAllPatients();

comboPatient.DataSource = business.PatientNames;

business.GetAllDentists();

comboDentist.DataSource = business.DentistNames;

comboPatient\_SelectedIndexChanged(sender, e);

}

Селектирайте бутона btnReserveAppointment и от Properties -> Events цъкнете двукратно върху Click. Автоматично се генерира празен метод btnReserveAppointment\_Click. Добавете код:

private void btnReserveAppointment\_Click(object sender, EventArgs e)

{

DateTime dt = dateAppointmentPicker.Value;

dt = dt.Date; // gets date part only, zeroes the time -> hh:mm:ss to 00:00:00

dt = dt.AddHours((double)numHour.Value);

dt = dt.AddMinutes((double)numMinutes.Value);

string appointmentStart = dt.ToString("yyyy-MM-dd HH:mm:ss");

DateTime dte = dt.AddMinutes((double)numDurationMinutes.Value);

string appointmentEnd = dte.ToString("yyyy-MM-dd HH:mm:ss");

long collisions = business.CheckAppointmentForCollisions(

business.DentistIDs[comboDentist.SelectedIndex],

appointmentStart, appointmentEnd);

if (collisions != 0)

{

MessageBox.Show("ERROR: Appointment collides with another one. \n Try a different day or hour.",

"ERROR: Appointmen failed", MessageBoxButtons.OK);

}

else

{

int added = business.AddAppointment(

business.PatientIDs[comboPatient.SelectedIndex],

business.DentistIDs[comboDentist.SelectedIndex],

appointmentStart, appointmentEnd);

if(added == 1)

MessageBox.Show("New appointment successfuly added.",

"Success", MessageBoxButtons.OK);

}

}

Този код чете дата и час от dateAppointmentPicker, numHour и numMinutes и ги форматира в appointmentStart, добавя минутите от numDurationMinutes и ги форматира в appointmentEnd. След това проверява за колизии в резервациите за текущо селектирания зъболекар с дадено начало и край на резервацията (CheckAppointmentForCollisions).

Ако има колизия се извежда съобщение за грешка.

Ако не - новата резервация се добавя в базата данни с AddAppointment.

------------------