ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

РАЗРАБОТКА ПРОГРАММЫ УЧЕТА УСПЕВАЕМОСТИ СТУДЕНТОВ

Курсовая работа

по дисциплине «Основы алгоритмизации и программирования»

КР.1-53 01 02.№10028412.№ 15

Исполнитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Самусев Д.А. АСОИ-181

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кашпар А. И.

(подпись)

Дата допуска к защите \_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты \_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_

Могилев 2019

ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Факультет Электротехнический

«Утверждаю»

Заведующий кафедрой\_\_\_\_\_\_\_\_\_\_

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

## З А Д А Н И Е

на курсовую работу по дисциплине «Основы алгоритмизации и программирования» студенту Самусев Д.А гр. № АСОИ-181

1. Тема курсовой работы: Разработка программы учёта успеваемости студентов

2. Срок сдачи законченной работы:

3. Исходные данные к курсовой работе:

Сведения об успеваемости студентов содержат следующую информацию: номер группы (шесть цифр), Ф.И.О. студента, сведения о пяти зачетах (зачет/незачет), оценки по пяти экзаменам. Индивидуальное задание: вывести всех студентов в порядке убывания количества задолженностей (количество задолженностей по каждому студенту необходимо указать); средний балл, полученный каждым студентом группы х (вводится с клавиатуры), и всей группой в целом.

4. Перечень подлежащих разработке вопросов:

Введение

1 Техническое задание

2 Описание программы

3 Руководство оператора

4 Тестирование

Заключение

Список использованных источников

5. На проверку предоставляются пояснительная записка, исходные тексты

программ и исполняемые файлы на электронном носителе.

Руководитель курсовой работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кашпар А. И.

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Самусев Д. А.

Содержание

Введение ................................................................................................................. 3

1 Техническое задание .......................................................................................... 4

2 Описание программы ......................................................................................... 5

2.1 Общие сведения ..................................................................................... 5

2.2 Функциональное назначение ................................................................ 5

2.3 Описание логической структуры ......................................................... 5

2.4 Используемые технические средства .................................................. 5

2.5 Вызов и загрузка .................................................................................... 5

2.6 Входные данные .................................................................................... 6

2.7 Выходные данные ................................................................................. 6

3 Руководство оператора ...................................................................................... 7

3.1 Назначение программы ........................................................................ 7

3.2 Условия выполнения программы ........................................................ 7

3.3 Выполнение программы ....................................................................... 7

3.4 Сообщение оператору ......................................................................... 14

Заключение ........................................................................................................... 15

Литература ........................................................................................................... 16

Приложение А....................................................................................................... 17

## Введение

## Целью выполнения данной работы является закрепление основ и углубление знаний приемов программирования на языке С#, получение практических навыков на всех этапах создания программного продукта: от постановки задачи до практической реализации и инструкциями по его использованию. В процессе выполнения работы необходимо решить следующие задачи: изучить и освоить паттерн проектирования MVC, внедрить HTML разметку в проект, предусмотреть возможные исключительные ситуации, разработать систему аутентификации и авторизации в приложении.

## В результате разработки данного приложения ожидается увидеть веб-сервис позволяющий контролировать успеваемость студентов конкретного учебного заведения. Содержание пояснительной записки: титульный лист, задание на курсовую работу, содержание, введение, основную часть, заключение, список использованных источников. Раздел «Приложение А» содержит код проекта.

## 1 Техническое задание

## Разработка программы учета успеваемости студентов. Сведения об успеваемости студентов содержат следующую информацию: номер группы (шесть цифр), Ф.И.О. студента, сведения о пяти зачетах (зачет/незачет), оценки по пяти экзаменам. Индивидуальное задание: вывести всех студентов в порядке убывания количества задолженностей (количество задолженностей по каждому студенту необходимо указать); средний балл, полученный каждым студентом группы х (вводится с клавиатуры), и всей группой в целом.

## 

## 2 Описание программы

2.1 Общие сведения

Программа предназначена для контроля успеваемости студентов. Реализована в среде Microsoft Visual Studio 2017 на языке программирования C#.

Автором проекта является студент группы АСОИ-181 Самусев Данила Андреевич. Проект разрабатывался с мая 2019 года в течении 2-ух месяцев.

Для корректной работы программы необходимаWindows 7 с установленным .NET Framework 4. Так же необходимы браузеры Opera/Chrome. Большая производительность аппаратного обеспечения не требуется. Так же необходима установленное и настроенное ПО для работы с базой данных MySql.

2.2 Функциональное назначение

Программа предназначена для контроля успеваемости студентов конкретного учебного заведения.

2.3 Описание логической структуры

Программа разрабатывалась на основе паттерна проектирования MVC (Model-View-Controller). Для взаимодействия с MySQL базой используется фрэймворк Dapper. Пользовательский интерфейс написан с помощью HTML, CSS, JavaScript. Приложение делится на модуль администратора и модуль пользователя. Модуль администратора включает в себя добавление, удаление и редактирование учётных записей а так же те же действия для основных данных программы с помощью графического интерфейса. Основные данные представлены в виде SQLфайла и расположены в корне программы. Пользователь или администратор могут использовать эти данные либо заполнить данные вручную.

2.4 Используемые технические средства

Программа эксплуатируется на персональном компьютере. Для работы с приложением используются мышь, клавиатура и дисплей монитора. Входные данные хранятся на жёстком диске в файле с расширением sql.

2.5 Вызов и загрузка

Так как данная программа является MVC приложением, то её запуск через файл с расширением exe довольно затруднителен. Поэтому приложение запускается с помощью IDE. При загрузке программы появляется вступительная форма с данными о создателе программы.

2.6 Входные данные

Входные данные находятся в Sql таблицах. Они создаются автоматические при запуске сервера.

2.7 Выходные данные

Взаимодействие с пользователем осуществляется за счёт написанного пользовательского интерфейса. Таким образом выходные данные являются HTML страницей. 3 Руководство оператора

3.1 Назначение программы

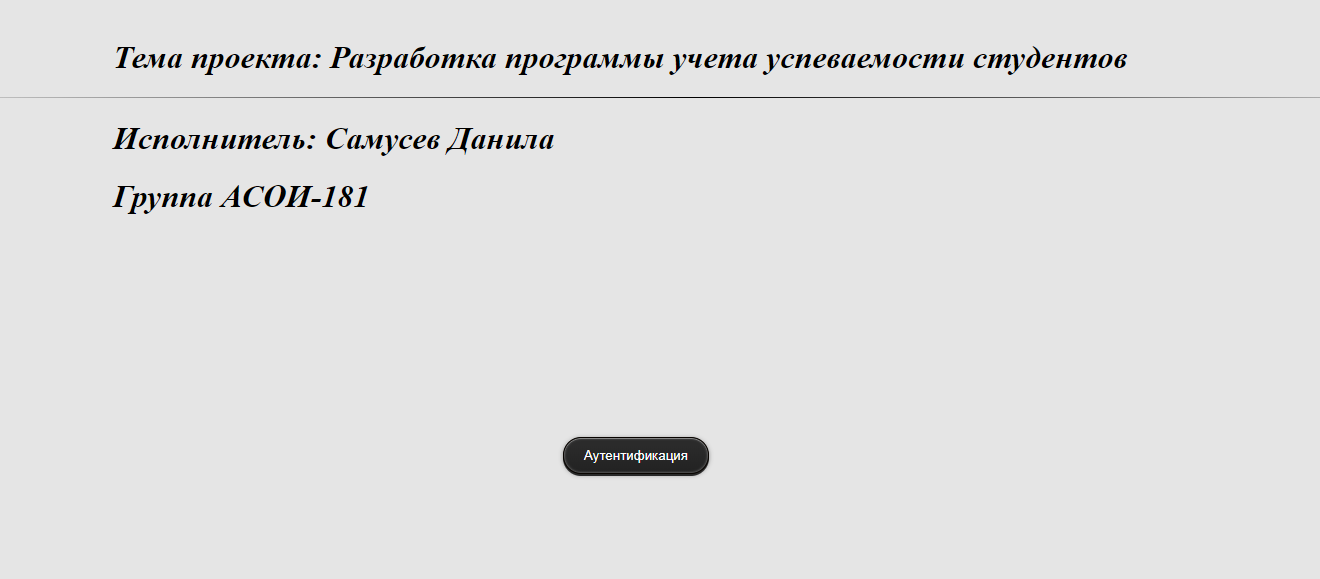
Назначением программы является контроль успеваемости студентов некоторого вуза ВУЗа.

3.2 Условия выполнения программы

Для выполнения программы необходимо соединение с базой данных MySql. Требуется наличие MySql Workbench 8.0 CE. Сервер настраивается на имя пользователя: root, сервер: localhost, пароль: 1234. Должна существовать база данных с именем database=student\_rating\_base.

3.3 Выполнение программы

Программа запускается с помощью IDE. При загрузке программы появляется вступительная форма с данными о создателе программы (рис.1).

 рисунок 1

При нажатии на кнопку "аутентификация" происходит переадресация на форму аутентификации(рис. 2).

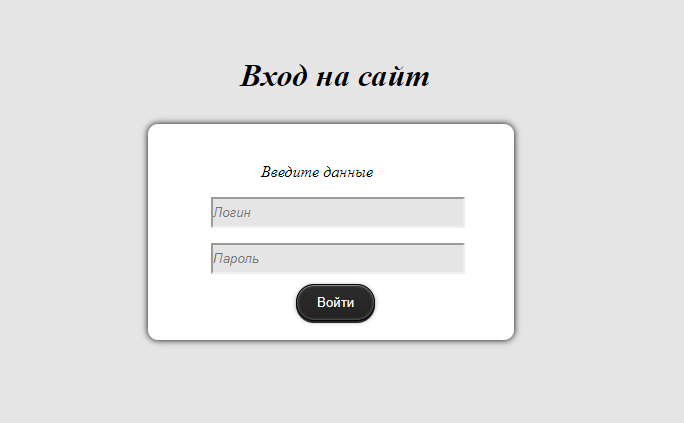


рисунок 2

Здесь вводятся данные пользователя. Если аутентификация прошла успешно, то происходит переход в меню (рис. 3).

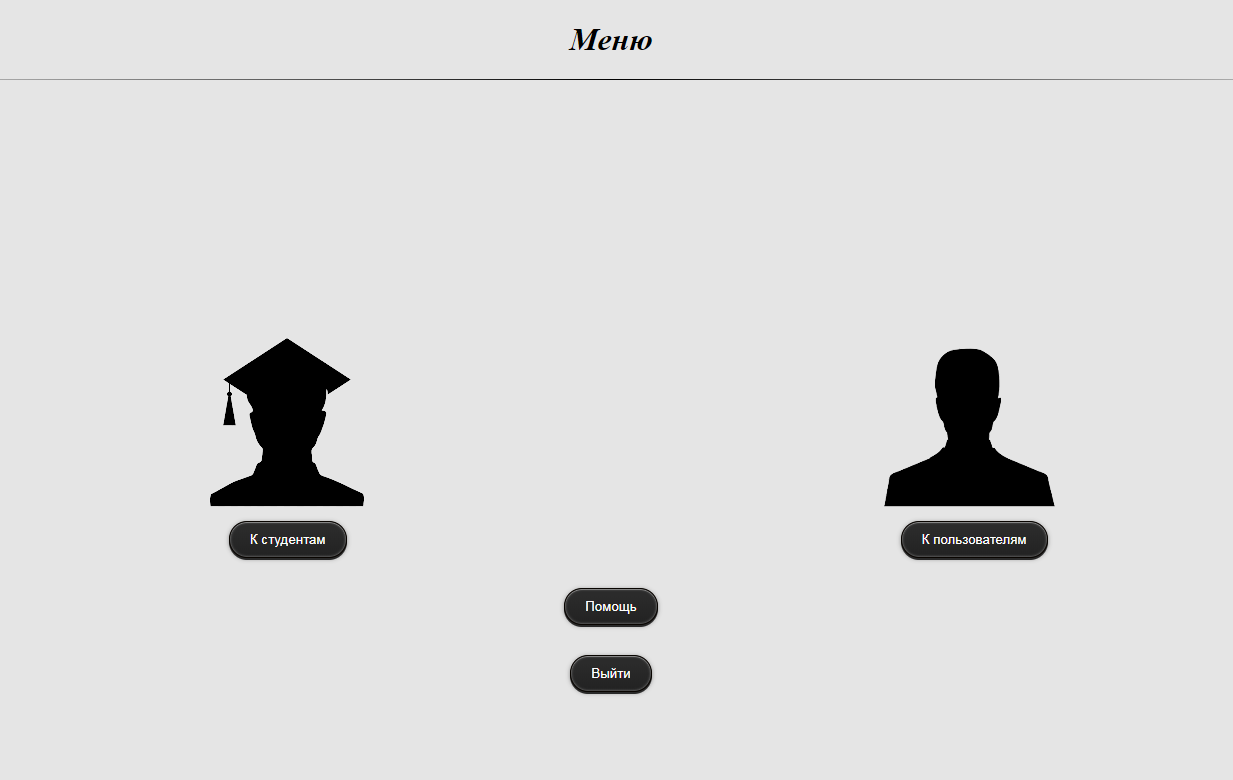


рисунок 3

Раздел меню делится на два блока: блок администратора и блок пользователя. При нажатии кнопки "к пользователям" происходит переход к странице для работы с пользователем (рис. 4). Примечание: при входе в качестве пользователя, блок работы с пользователями скрыт.

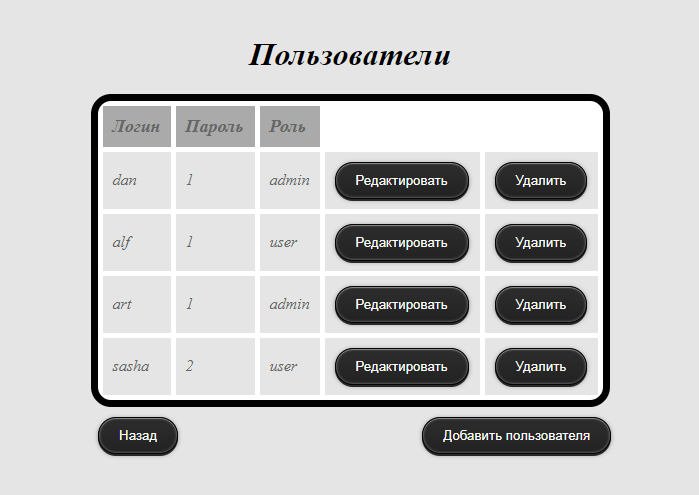


рисунок 4

На данной странице можно редактировать, удалять и добавлять нового пользователя. Для этого предусмотрены всплывающие окна, либо отдельные страницы для более сложной логики работы (рис. 5).

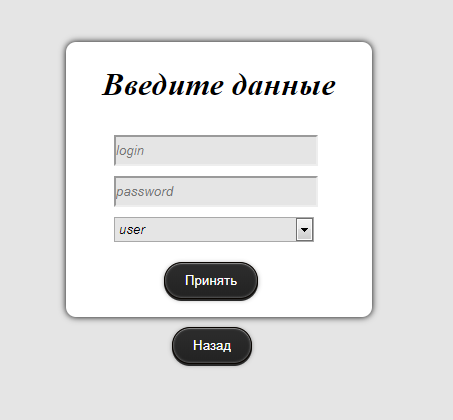


рисунок 5

При нажатии кнопки к студентам происходит переход к странице для работы со студентами. Студенты по заданию отсортированы по количеству академических задолжностей по возрастанию (рис. 6).



рисунок 6

На данной странице можно редактировать, удалять и добавлять а так же производить поиск студентов. Для этого предназначены всплывающие окна, либо отдельные странички для более сложной логики (рис. 7). Для поиска студентов по особым критериям используется форма поиска под таблицей.

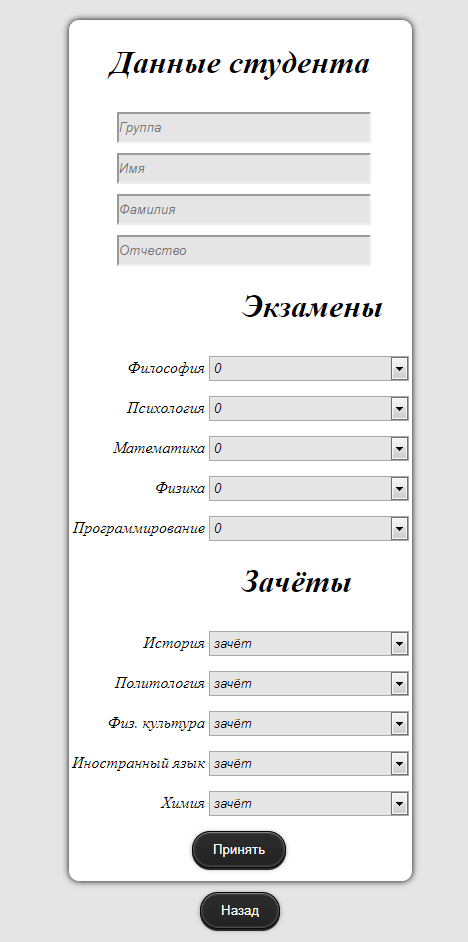


рисунок 7

Так же можно подробнее просмотреть и отредактировать данные студента по пяти экзаменам и пяти зачётам (рис. 8), (рис. 9) соответственно.

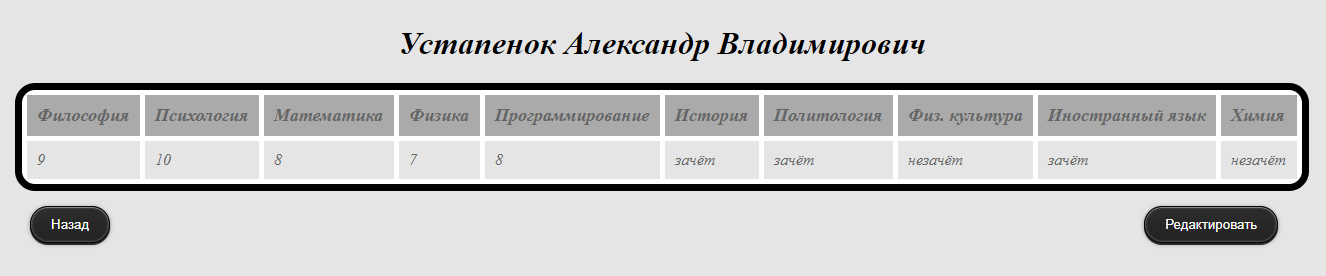


рисунок 8

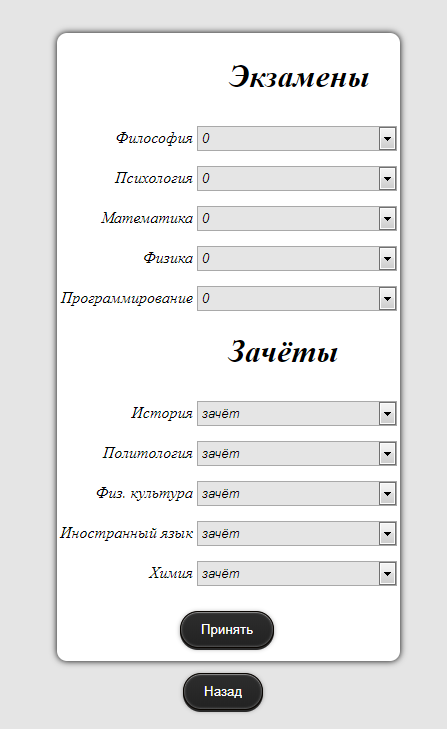


рисунок 9

После выполнения какой-либо важной операции на сайте, программа оповещает пользователя об этом (рис. 10).

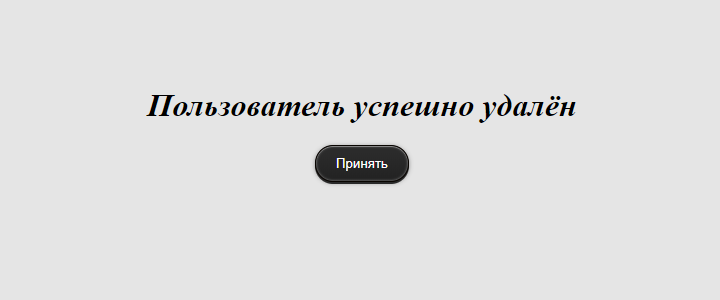


рисунок 10

3.4 Сообщения оператору

Сообщения об ошибках или результате работы выводятся на специальную HTML страницу.

Заключение

## В результате выполнения данной работы были закрепление основы и углублены знания приемов программирования на языке С#, получены практические навыки на всех этапах создания программного продукта. В процессе выполнения работы решены следующие задачи: изучение и освоение паттерна проектирования MVC, внедрениеHTMLразметки в проект, предусмотрены возможные исключительные ситуации, разработана система аутентификации и авторизации в приложении.

В итоге был разработан веб-сервис позволяющий контролировать успеваемость студентов конкретного учебного заведения.

Литература

1. http://htmlbook.ru

2. https://docs.microsoft.com

3. http://professorweb.ru

4. https://metanit.com/sharp

5.https://stackoverflow.com

Приложение А

Startup.cs

**namespace** AccountingSystem

{ **public class** Startup  
 {  
   
 **public** IConfiguration Configuration { get; }  
   
 **public** Startup(IConfiguration configuration)  
 {  
 Configuration = configuration;   
   
 **using** (MySqlConnection connection = **new** MySqlConnection("server=localhost;user=root;database=student\_rating\_base;password=1234;"))  
 {   
 StreamReader reader1 = **new** StreamReader("wwwroot/sql/tableCreationScript.sql");  
 StreamReader reader2 = **new** StreamReader("wwwroot/sql/tablesFillingScript.sql");  
 **string** script1 = reader1.ReadToEnd();  
 **string** script2 = reader2.ReadToEnd();  
 connection.Open();  
 connection.Query(script1);  
 connection.Query(script2);  
 }  
 }  
  
 // This method gets called by the runtime. Use this method to add services to the container.  
 **public void** ConfigureServices(IServiceCollection services)  
 {   
   
 **string** connectionString = Configuration.GetConnectionString("ConnectionString");  
  
 services.AddSession();  
 services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.*Version\_2\_2*).AddFluentValidation();  
 services.AddSingleton<IUserRepository, UserRepository>(ur => **new** UserRepository(connectionString));  
 services.AddSingleton<IStudentRepository, StudentRepository>(sr => **new** StudentRepository(connectionString));  
 services.AddSingleton<IRatingRepository, RatingRepository>(rr => **new** RatingRepository(connectionString));  
 services.AddSingleton<Validator>();  
 services.AddSingleton<AbstractValidator<ExamsRating>, ExamsRatingValidator>();  
 services.AddSingleton<AbstractValidator<ScoresRating>, ScoresRatingValidator>();  
 services.AddSingleton<AbstractValidator<Student>, StudentValidator>();  
 services.AddSingleton<AbstractValidator<LoginModel>, LoginModelValidator>();  
 services.AddSingleton<AbstractValidator<User>, UserValidator>();  
 }  
  
 // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.  
 **public void** Configure(IApplicationBuilder app, IHostingEnvironment env)  
 {  
   
 **if** (env.IsDevelopment())  
 {   
 app.UseDeveloperExceptionPage();  
 }  
  
 app.UseSession();  
 app.UseStaticFiles();  
 app.UseHttpsRedirection();   
   
 app.UseMvc(routes =>  
 {  
 routes.MapRoute(  
 name: "default",  
 template: "{controller=Preview}/{action=Preview}/{id?}");  
 });  
 }  
 }

}

UserController.cs

**namespace** AccountingSystem.Controllers

{ **public class** UsersController : Controller  
 {  
 **private readonly** IUserRepository \_userRepository;  
 **private readonly** Validator \_validator;  
 **private readonly** AbstractValidator<User> \_userValidator;  
 **private const string** *Users\_* = "users";  
 **private const string** *UserExists* = "Пользователь с таким логином уже существует!";  
  
 **public** UsersController(IUserRepository userRepository, Validator validator,  
 AbstractValidator<User> userValidator)  
 {  
 \_userRepository = userRepository;  
 \_validator = validator;  
 \_userValidator = userValidator;  
 }  
  
 **public** IActionResult Users()  
 {  
 IActionResult result = CheckUserAccess();  
   
 **if** (result != **null**)  
 {  
 **return** result;  
 }  
   
 List<User> users = GetUsers();  
   
 **return** View(users);  
 }  
  
 **public** IActionResult DeleteUser(**long** userId)  
 {   
 IActionResult result = CheckUserAccess();  
   
 **if** (result != **null**)  
 {  
 **return** result;  
 }  
  
 **if** (userId < 0)  
 {  
 **return** View("~/Views/Error400.cshtml");  
 }  
   
 List<User> users = GetUsers();  
  
 \_userRepository.DeleteOneById(userId);  
 User user = users.FirstOrDefault(u => u.Id == userId);  
 users.Remove(user);  
 HttpContext.Session.Set(*Users\_*, users);  
  
 **return** RedirectToAction("UsersResult", "Users", **new** {message = "Пользователь успешно удалён"});  
 }  
  
 **public** IActionResult ModifyUser(User user)  
 {   
 IActionResult actionResult = CheckUserAccess();  
   
 **if** (actionResult != **null**)  
 {  
 **return** actionResult;  
 }  
   
 ValidationResult result = \_userValidator.Validate(user);  
  
 **if** (!result.IsValid)  
 {  
 **return** View("~/Views/Error400.cshtml");  
 }  
   
 List<User> users = GetUsers();  
  
 **if** (\_validator.UserIsUnique(user ,users))  
 {  
 User oldUser = users.FirstOrDefault(u => u.Id == user.Id);  
  
 \_userRepository.Modify(user);  
 users.Remove(oldUser);  
 users.Add(user);  
 HttpContext.Session.Set(*Users\_*, users);   
 HttpContext.Session.Set("userModifyError", "");  
 }  
 **else** {  
 **string** modifyError = *UserExists*;  
 HttpContext.Session.Set("userModifyError", modifyError);  
 }  
   
 **return** RedirectToAction("UsersResult", "Users", **new** {message = "Пользователь успешно обновлён"});  
 }  
  
 [HttpGet]  
 **public** IActionResult AddUser()  
 {   
 IActionResult result = CheckUserAccess();  
   
 **if** (result != **null**)  
 {  
 **return** result;  
 }  
   
 HttpContext.Session.Set("userAddError", "");   
 **return** View();  
 }  
   
 [HttpPost]  
 **public** IActionResult AddUser(User user)  
 {  
 IActionResult actionResult = CheckUserAccess();  
   
 **if** (actionResult != **null**)  
 {  
 **return** actionResult;  
 }  
  
 ValidationResult result = \_userValidator.Validate(user);  
  
 **if** (!result.IsValid)  
 {  
 **return** View("~/Views/Error400.cshtml");  
 }  
   
 List<User> users = GetUsers();  
  
 **if** (\_validator.UserIsUnique(user ,users))  
 {  
 \_userRepository.Add(user);  
 users.Add(user);  
 HttpContext.Session.Set(*Users\_*, users);  
 HttpContext.Session.Set("userAddError", "");   
 }  
 **else** {  
 **string** userAddError = *UserExists*;  
 HttpContext.Session.Set("userAddError", userAddError);  
 **return** View();  
 }  
  
 **return** RedirectToAction("UsersResult", "Users", **new** {message = "Пользователь успешно добавлен"});  
 }  
  
 **public** IActionResult UsersResult(**string** message)  
 {  
 **return** View(model: message);  
 }  
   
 **private** List<User> GetUsers()  
 {  
 List<User> users = HttpContext.Session.Get<List<User>>(*Users\_*) ?? \_userRepository.GetAll();  
  
 **return** users;  
 }  
  
 **public** IActionResult CheckUserAccess()  
 {  
 User currentUser = HttpContext.Session.Get<User>("user");  
  
 **if** (currentUser.Role != "admin")  
 {  
 **return** View("AccessError");  
 }  
  
 **return null**;  
 }  
 }

}

UserRepository.cs

**namespace** AccountingSystem.Repository

{ **public class** UserRepository : IUserRepository  
 {  
  
 **private string** ConnectionString { get; set; }  
   
 **public** UserRepository(**string** connectionString)  
 {  
 ConnectionString = connectionString;  
 }  
  
 **public** List<User> GetAll()  
 {  
 **using** (MySqlConnection connection = **new** MySqlConnection(ConnectionString))  
 {  
 **string** strQuery = "Select \* from users";  
  
 List<User> users = connection.Query<User>(strQuery).ToList();  
  
 **return** users;  
 }  
 }  
   
 **public** User GetOneByLoginAndPassword(**string** login, **int** password)  
 {  
 **using** (MySqlConnection connection = **new** MySqlConnection(ConnectionString))  
 {  
 **string** strQuery = "Select \* from users" +  
 " where login = @login and password = @password";  
  
 User user = connection.Query<User>(strQuery, **new** {login, password}).FirstOrDefault();  
  
 **return** user;  
 }   
 }  
   
 **public void** Modify(User user)  
 {   
 **using** (MySqlConnection connection = **new** MySqlConnection(ConnectionString))  
 {  
 **string** strQuery = "Update users set" +  
 " login = @Login," +  
 " password = @Password," +  
 " role = @Role" +  
 " where id = @Id";  
  
 connection.Query(strQuery, user);  
 }  
 }  
  
 **public void** Add(User user)  
 {   
 **using** (MySqlConnection connection = **new** MySqlConnection(ConnectionString))  
 {  
 **string** strQuery = "Insert into users" +  
 " (login, password, role)" +  
 " values" +  
 " (@Login, @Password, @Role)";  
  
 connection.Query(strQuery, user);  
 }  
 }  
   
 **public void** DeleteOneById(**long** userId)  
 {  
 **using** (MySqlConnection connection = **new** MySqlConnection(ConnectionString))  
 {  
 **string** strQuery = "Delete from users" +  
 " where id = @userId";  
  
 connection.Query(strQuery, **new** {userId});   
 }  
 }  
   
 }

}

StudentController.cs

**namespace** AccountingSystem.Controllers

{ **public class** StudentsController : Controller  
 {  
 **private readonly** IStudentRepository \_studentRepository;  
 **private readonly** IRatingRepository \_ratingRepository;  
 **private readonly** AbstractValidator<ExamsRating> \_examsRatingValidator;  
 **private readonly** AbstractValidator<ScoresRating> \_scoresRatingValidator;  
 **private readonly** AbstractValidator<Student> \_studentValidator;  
 **private const string** *Students\_* = "students";  
  
 **public** StudentsController(IStudentRepository studentRepository, IRatingRepository ratingRepository,  
 Validator validator,  
 AbstractValidator<ExamsRating> examsRatingValidator, AbstractValidator<ScoresRating> scoresRatingValidator,  
 AbstractValidator<Student> studentValidator)  
 {  
 \_studentRepository = studentRepository;  
 \_ratingRepository = ratingRepository;  
 \_examsRatingValidator = examsRatingValidator;  
 \_scoresRatingValidator = scoresRatingValidator;  
 \_studentValidator = studentValidator;  
 }  
  
 **public** IActionResult Students()  
 {  
 List<Student> students = GetStudents();  
   
 //////  
 List<Student> requiredStudents = HttpContext.Session.Get<List<Student>>("requiredStudents");  
 **bool** searchIsActive = HttpContext.Session.Get<**bool**>("searchIsActive");  
 //////  
   
 **if** (requiredStudents == **null** || requiredStudents.Count == 0)  
 {  
 **if** (searchIsActive)  
 {   
 ResetSearch();  
 **return** RedirectToAction("StudentsResult", "Students",  
 **new** {message = "По вашему запросу ничего не найдено"});  
 }  
 }  
 **else** {  
 students = requiredStudents;   
 }  
   
 FillRating(students);  
 students.Sort();  
 **return** View(students);  
 }  
  
 **public** IActionResult DeleteStudent(**long** id)  
 {  
 **if** (id < 0)  
 {  
 **return** View("~/Views/Error400.cshtml");  
 }  
  
 List<Student> students = GetStudents();  
 Student student = students.FirstOrDefault(s => s.Id == id);  
  
 \_studentRepository.Delete(student);  
 students.Remove(student);  
 HttpContext.Session.Set(*Students\_*, students);  
 ResetSearch();  
   
 **return** RedirectToAction("StudentsResult", "Students",  
 **new** {message = "Студент успешно удалён"});  
 }  
  
 **public** IActionResult UpdateStudent(Student student)  
 {  
 ValidationResult validationResult = \_studentValidator.Validate(student);  
  
 **if** (!validationResult.IsValid)  
 {  
 **return** View("~/Views/Error400.cshtml");   
 }  
   
 List<Student> students = GetStudents();  
  
 Student oldStudent = students.FirstOrDefault(s => s.Id == student.Id);  
  
 \_studentRepository.Modify(student);  
 students.Remove(oldStudent);  
 students.Add(student);  
 HttpContext.Session.Set(*Students\_*, students);  
  
 ResetSearch();  
   
 **return** RedirectToAction("StudentsResult", "Students",  
 **new** {message = "Студент успешно обновлен"});  
 }  
  
 [HttpGet]  
 **public** IActionResult AddStudent()  
 {  
 **return** View();  
 }  
  
 [HttpPost]  
 **public** IActionResult AddStudent(Student student, ExamsRating examsRating, ScoresRating scoresRating)  
 {  
 List<Student> students = GetStudents();  
  
 ValidationResult examValidationResult = \_examsRatingValidator.Validate(examsRating);  
 ValidationResult scoreValidationResult = \_scoresRatingValidator.Validate(scoresRating);  
 ValidationResult studetnValidationResult = \_studentValidator.Validate(student);  
  
 **if** (!examValidationResult.IsValid || !scoreValidationResult.IsValid || !studetnValidationResult.IsValid)  
 {  
 **return** View("~/Views/Error400.cshtml");  
 }  
  
 student = \_studentRepository.Add(student);  
 examsRating.StudentId = student.Id;  
 scoresRating.StudentId = student.Id;  
 \_ratingRepository.AddExamRating(examsRating);  
 \_ratingRepository.AddScoreRating(scoresRating);   
 students.Add(student);  
 HttpContext.Session.Set(*Students\_*, students);  
  
 ResetSearch();  
   
 **return** RedirectToAction("StudentsResult", "Students",  
 **new** {message = "Студент успешно добавлен"});  
 }  
  
 **public** IActionResult StudentsResult(**string** message)  
 {  
 **return** View(model: message);  
 }  
  
 **private void** FillRating(List<Student> students)  
 {  
 **foreach** (Student student **in** students)  
 {  
 **int** debts = 0;  
  
 IDictionary<**string**, **object**> examsRating = \_ratingRepository.GetExamsRating(student.Id);  
 IDictionary<**string**, **object**> scoresRating = \_ratingRepository.GetScoresRating(student.Id);  
  
 **int** examsDebts = examsRating.Values.Count(ex => (**int**) ex < 4);  
 **int** scoresDebts = scoresRating.Values.Count(sc => sc.Equals("незачёт"));  
  
 debts += examsDebts + scoresDebts;  
 student.Debts = debts;  
 }  
 }  
  
 **public** IActionResult Search(**string** option, **string** value)  
 {  
 List<Student> requiredStudents;  
 List<Student> students = GetStudents();  
  
 **switch** (option)  
 {  
 **case** "sortByDebts":  
 {  
 **int** debts;  
  
 **try** {  
 debts = **int**.Parse(value);  
 }  
 **catch** {   
 ResetSearch();  
 **return** RedirectToAction("StudentsResult", "Students",  
 **new** {message = "Некорректные данные"});  
 }  
  
 FillRating(students);  
 requiredStudents = students.Where(s => s.Debts == debts).ToList();  
  
 **return** SetSearch(requiredStudents);  
 }  
 **case** "sortByGroup":  
 {  
 **int** groupNumber;  
  
 **try** {  
 groupNumber = **int**.Parse(value);  
 }  
 **catch** {  
 ResetSearch();  
 **return** RedirectToAction("StudentsResult", "Students",  
 **new** {message = "Некорректные данные"});  
 }  
  
 requiredStudents = students.Where(s => s.Group\_Number == groupNumber).ToList();  
  
 **return** SetSearch(requiredStudents);  
 }  
 **case** "sortBySurname":  
 {  
 **string** surname = value;  
 requiredStudents = students.Where(s => s.Surname == surname).ToList();  
  
 **return** SetSearch(requiredStudents);  
 }  
 }  
  
 **return** RedirectToAction("Students", "Students");  
 }  
  
 **private** IActionResult SetSearch(List<Student> requiredStudents)  
 {  
 HttpContext.Session.Set("requiredStudents", requiredStudents);  
 HttpContext.Session.Set("searchIsActive", **true**);  
 **return** RedirectToAction("Students", "Students");  
 }  
  
 **private void** ResetSearch()  
 {  
 HttpContext.Session.Set("requiredStudents", **new** List<Student>());  
 HttpContext.Session.Set("searchIsActive", **false**);   
 }   
  
 **private** List<Student> GetStudents()  
 {  
 List<Student> students = HttpContext.Session.Get<List<Student>>(*Students\_*) ?? \_studentRepository.GetAll();  
  
 **return** students;  
 }  
 }

}

RatingController.cs

**namespace** AccountingSystem.Controllers

{ **public class** RatingController : Controller  
 {  
 **private readonly** IStudentRepository \_studentRepository;  
 **private readonly** IRatingRepository \_ratingRepository;  
 **private readonly** AbstractValidator<ExamsRating> \_examsRatingValidator;  
 **private readonly** AbstractValidator<ScoresRating> \_scoresRatingValidator;  
 **private const string** *Students\_* = "students";  
   
 **public** RatingController(IRatingRepository ratingRepository, IStudentRepository studentRepository,  
 AbstractValidator<ExamsRating> examsValidator, AbstractValidator<ScoresRating> scoresRatingValidator)  
 {  
 \_ratingRepository = ratingRepository;  
 \_studentRepository = studentRepository;  
 \_examsRatingValidator = examsValidator;  
 \_scoresRatingValidator = scoresRatingValidator;  
 }  
   
 **public** IActionResult Rating(**long** studentId)  
 {  
 **if** (studentId < 0)  
 {  
 **return** View("~/Views/Error400.cshtml");  
 }  
   
 List<Student> students = GetStudents();  
 Student currentStudent = students.FirstOrDefault(s => s.Id == studentId);  
   
 HttpContext.Session.Set("currentStudent", currentStudent);  
 IDictionary<**string**, **object**> examsRating = \_ratingRepository.GetExamsRating(currentStudent.Id);  
 IDictionary<**string**, **object**> scoresRating = \_ratingRepository.GetScoresRating(currentStudent.Id);   
 HttpContext.Session.Set("examsRating", examsRating);  
 HttpContext.Session.Set("scoresRating", scoresRating);  
   
 **return** View();  
 }  
  
 [HttpGet]  
 **public** IActionResult ModifyRating(**long** studentId)  
 {   
 HttpContext.Session.Set("currentStudentId", studentId);  
 **return** View();  
 }  
   
 [HttpPost]  
 **public** IActionResult ModifyRating(ExamsRating examsRating, ScoresRating scoresRating)  
 {  
 ValidationResult examValidationResult = \_examsRatingValidator.Validate(examsRating);  
 ValidationResult scoreValidationResult = \_scoresRatingValidator.Validate(scoresRating);  
  
 **if** (!examValidationResult.IsValid || !scoreValidationResult.IsValid)  
 {  
 **return** View("~/Views/Error400.cshtml");  
 }  
   
 \_ratingRepository.Modify(examsRating, scoresRating);  
  
 **return** View("Result");   
 }  
   
 **private** List<Student> GetStudents()  
 {  
 List<Student> students = HttpContext.Session.Get<List<Student>>(*Students\_*) ?? \_studentRepository.GetAll();  
  
 **return** students;  
 }  
 }

}