Министерство науки и высшего образования Российской Федерации

Казанский национальный исследовательский технический университет – КАИ им. А.Н. Туполева

Институт компьютерных технологий и защиты информации

Отделение СПО ИКТЗИ «Колледж информационных технологий»

**ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ**

Методические указания к лабораторной работе №12

Тема: «Работа с EF Core 7 и C#»

Казань 2022

Методические указания к лабораторным работам по дисциплине «ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ» предназначены для студентов направления подготовки 09.02.07 «Информационные системы и программирование»

КНИТУ-КАИ, 2023

**ПРОЦЕСС СДАЧИ ВЫПОЛНЕННОЙ РАБОТЫ**

По итогам выполнения работы студент:

1. демонстрирует преподавателю правильно работающие программы;
2. демонстрирует приобретённые знания и навыки отвечает на пару небольших вопросов преподавателя по составленной программе, возможностям её доработки;
3. демонстрирует отчет по выполненной лабораторной работе.

Итоговая оценка складывается из оценок по трем указанным составляющим.

**ЛАБОРАТОРНАЯ РАБОТА №12**

**ТЕМА: «РАБОТА С ENTITY FRAMEWORK CORE 7 И C#»**

ЦЕЛЬ РАБОТЫ

Получить практические навыки работы с EF Core 7 и C# для реализации взаимодействия с базой данных.

ХОД РАБОТЫ

**Шаг 1. Создание проекта**

Создадим новый проект в Visual Studio, выбрав шаблон проекта "Windows Forms App (.NET Core)".

**Шаг 2. Добавление Entity Framework Core**

Добавим следующие пакеты Entity Framework Core, через «Управление пакетами NuGet»:

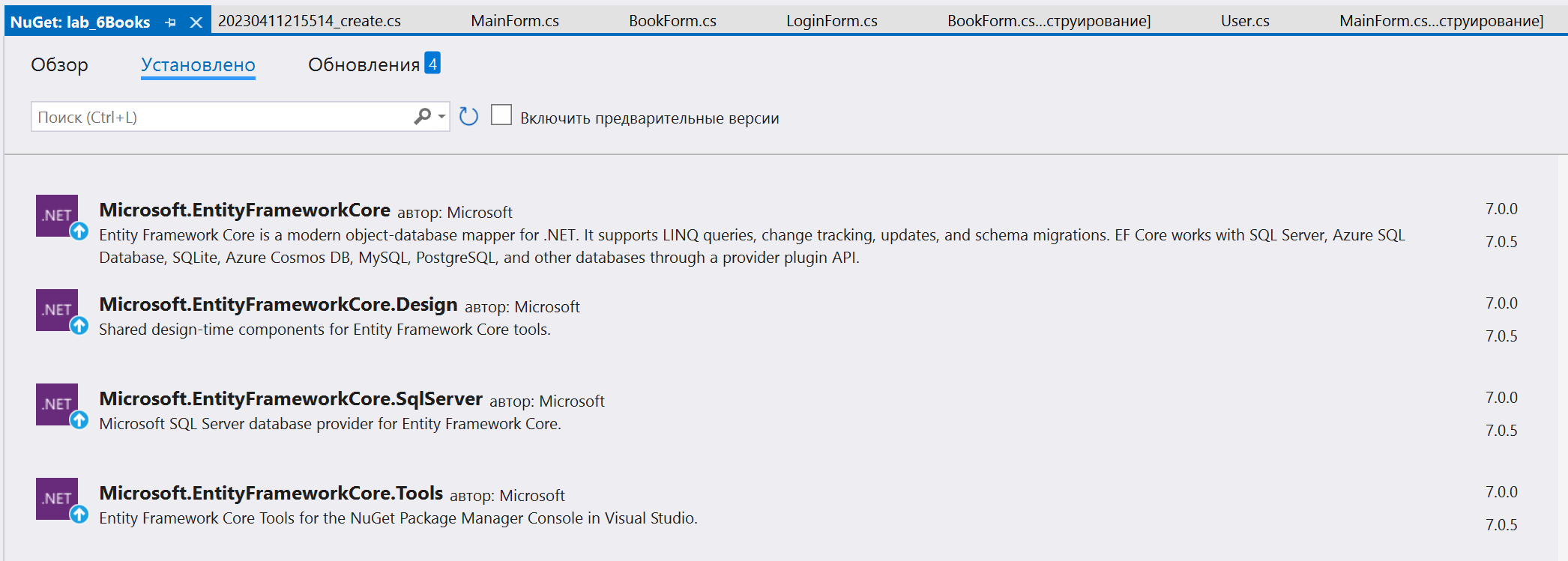


Рисунок 1 – Добавление необходимых пакетов

**Шаг 3. Создание модели данных**

Создадим модель данных, которую мы будем использовать для создания таблиц в базе данных. Для этого создадим классы, описывающие сущности и связи между ними.

|  |
| --- |
| public class User  {  public int Id { get; set; }  public string Name { get; set; }  public string Email { get; set; }  public string Password { get; set; }  public List<Book> Books { get; set; }  public User()  {  Books = new List<Book>();  }  } |

|  |
| --- |
| public class Book  {  public int Id { get; set; }  public string Title { get; set; }  public int Year { get; set; }  public string Author { get; set; }  public int UserId { get; set; }  public User User { get; set; }  } |

Здесь мы создали две сущности - User и Book, где каждый пользователь может иметь несколько книг. Связь между таблицами будет установлена по полю UserId.

**Шаг 4. Настройка подключения к базе данных**

Добавим класс, который будет использоваться для настройки подключения к базе данных.

|  |
| --- |
| public class AppDbContext : DbContext  {  private const string ConnectionString = "Data Source=(localdb)\\mssqllocaldb;Initial Catalog=LibraryDb;Integrated Security=True;";  public DbSet<User> Users { get; set; }  public DbSet<Book> Books { get; set; }  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  {  optionsBuilder.UseSqlServer(ConnectionString);  }  protected override void OnModelCreating(ModelBuilder modelBuilder)  {  modelBuilder.Entity<User>()  .HasMany(u => u.Books)  .WithOne(b => b.User)  .HasForeignKey(b => b.UserId);  }  } |

В методе OnConfiguring мы устанавливаем подключение к базе данных, используя строку подключения. В методе OnModelCreating мы устанавливаем связь между таблицами. Данный фрагмент кода относится к Fluent API в Entity Framework Core и определяет связь между двумя сущностями User и Book.

Метод **HasMany** указывает, что у сущности User может быть много книг, т.е. один пользователь может иметь множество книг.

Метод **WithOne** указывает, что каждая книга принадлежит только одному пользователю.

Метод **HasForeignKey** указывает внешний ключ для связи между двумя сущностями. В данном случае, в таблице Book будет создан столбец **UserId**, который будет ссылаться на первичный ключ в таблице User.

Таким образом, данный код определяет связь один-ко-многим между сущностями User и Book по полю UserId в таблице Book.

Остается лишь создать базу данных и таблицы, для этого воспользуемся миграциями.

**Миграции** — это набор изменений схемы базы данных, которые можно применять и откатывать при необходимости.

1. Для создания миграции нужно выполнить следующую команду в консоли диспетчера пакетов (Вид -> Другие окна -> Консоль диспетчера пакетов):

|  |
| --- |
| add-migration create |

Где «create» является лишь названием конкретной миграции, вы можете указать иное название.

1. Применяем миграции к базе данных. Выполняем следующую команду:

|  |
| --- |
| update-database |

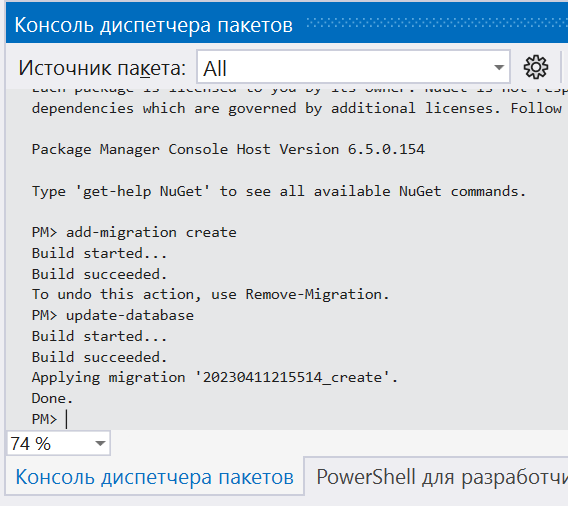


Рисунок 2 - Консоль диспетчера пакетов

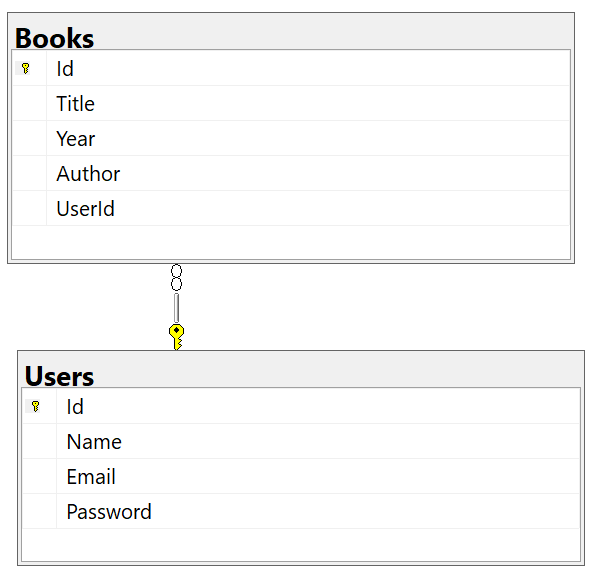


Рисунок 3 - Диаграмма таблиц

**Шаг 5. Создание форм для регистрации и авторизации пользователей**

Создадим форму для регистрации новых пользователей. Добавим текстовые поля для ввода имени, email/логина и пароля, а также кнопку "Зарегистрироваться".

|  |
| --- |
| public partial class RegistrationForm : Form  {  private readonly AppDbContext context;  public RegistrationForm(AppDbContext context)  {  InitializeComponent();  this.context = context;  }  private void RegisterButton\_Click(object sender, EventArgs e)  {  var user = new User  {  Name = NameTextBox.Text,  Email = EmailTextBox.Text,  Password = PasswordTextBox.Text  };  context.Users.Add(user);  context.SaveChanges();  MessageBox.Show("Вы успешно зарегистрировались!", "Регистрация", MessageBoxButtons.OK, MessageBoxIcon.Information);  Close();  }  } |

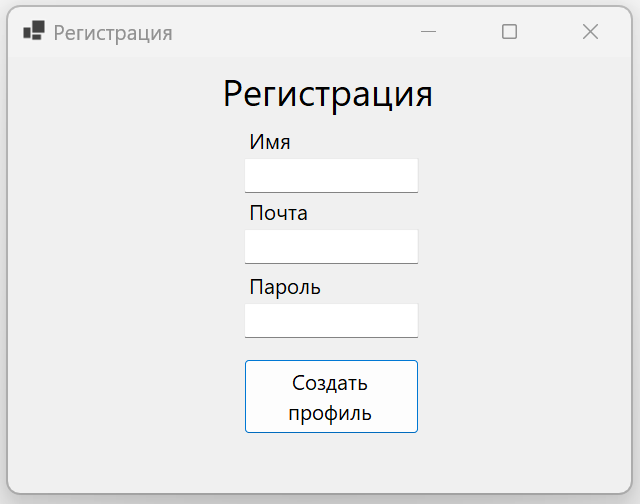


Рисунок 4 - Форма регистрации

Здесь мы создаем нового пользователя с данными, введенными пользователем, и сохраняем его в базе данных. После успешной регистрации выводим сообщение об успехе и закрываем форму.

Также создадим форму для авторизации пользователей. Добавим текстовые поля для ввода email и пароля, а также кнопку "Войти".

|  |
| --- |
| public partial class LoginForm : Form  {  public LoginForm()  {  InitializeComponent();  }  private void LoginButton\_Click(object sender, EventArgs e)  {  User user;  using (var context = new AppDbContext())  {  var email = EmailTextBox.Text;  var password = PasswordTextBox.Text;  user = context.Users.FirstOrDefault(u => u.Email == email && u.Password == password);  }  if (user != null)  {  MessageBox.Show("Вы успешно авторизовались!", "Авторизация", MessageBoxButtons.OK, MessageBoxIcon.Information);  MainForm mainForm = new MainForm(new AppDbContext(), user);  Hide();  mainForm.ShowDialog();  DialogResult dialogResult = MessageBox.Show("Закрыть программу?", "Думай", MessageBoxButtons.YesNo);  if (dialogResult == DialogResult.Yes)  Close();  else if (dialogResult == DialogResult.No)  Show();  }  else  {  MessageBox.Show("Неправильный email или пароль", "Ошибка авторизации", MessageBoxButtons.OK, MessageBoxIcon.Error);  }  }  private void RegistrationButton\_Click(object sender, EventArgs e)  {  RegistrationForm registrationForm = new RegistrationForm(new AppDbContext());  Hide();  registrationForm.ShowDialog();  Show();  }  } |

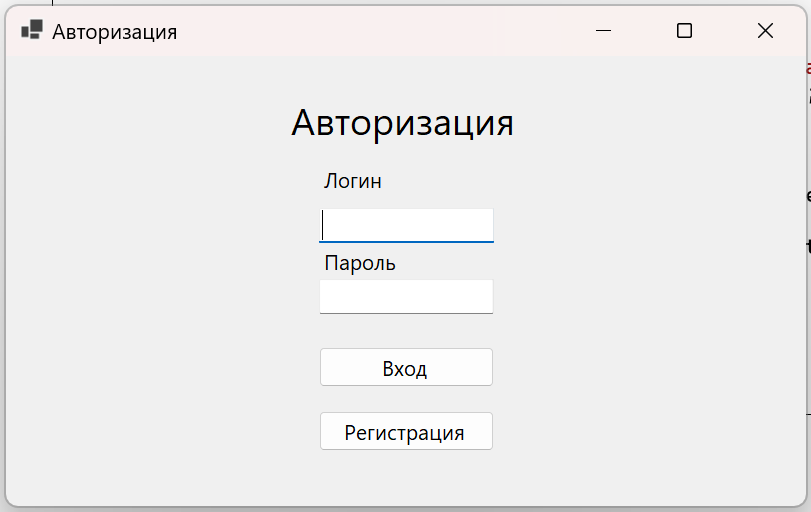


Рисунок 5 - Форма авторизации

Здесь мы проверяем, есть ли в базе данных пользователь с таким email/логином и паролем. Если пользователь найден, выводим сообщение об успехе и закрываем форму. Если пользователь не найден, выводим сообщение об ошибке.

**Шаг 6. Создание главной формы приложения**

Создадим главную форму приложения, на которой будут отображаться все книги, добавленные пользователями. Добавим таблицу для отображения списка книг, а также кнопки для добавления и удаления книг.

|  |
| --- |
| public partial class MainForm : Form  {  private AppDbContext context;  private User user;  public MainForm(AppDbContext context, User user)  {  InitializeComponent();  this.context = context;  this.user = user;  user.Books = context.Books.Where(b => b.UserId == user.Id).ToList();  Text = $"Библиотека пользователя {user.Name}";  BooksDataGridView.AutoGenerateColumns = true;  BooksDataGridView.DataSource = user.Books;  } |

Добавим обработчики событий для кнопок "Добавить книгу" и "Удалить книгу", а также для кнопки выхода.

|  |
| --- |
| private void AddBookButton\_Click(object sender, EventArgs e)  {  var bookForm = new BookForm(context, user);  if (bookForm.ShowDialog() == DialogResult.OK)  {  user.Books.Add(bookForm.Book);  context.SaveChanges();  BooksDataGridView.DataSource = null;  BooksDataGridView.DataSource = user.Books;  }  }  private void DeleteBookButton\_Click(object sender, EventArgs e)  {  var selectedRow = BooksDataGridView.SelectedRows[0];  var book = (Book)selectedRow.DataBoundItem;  user.Books.Remove(book);  context.SaveChanges();  BooksDataGridView.DataSource = null;  BooksDataGridView.DataSource = user.Books;  }  private void ExitButton\_Click(object sender, EventArgs e)  {  Close();  } |

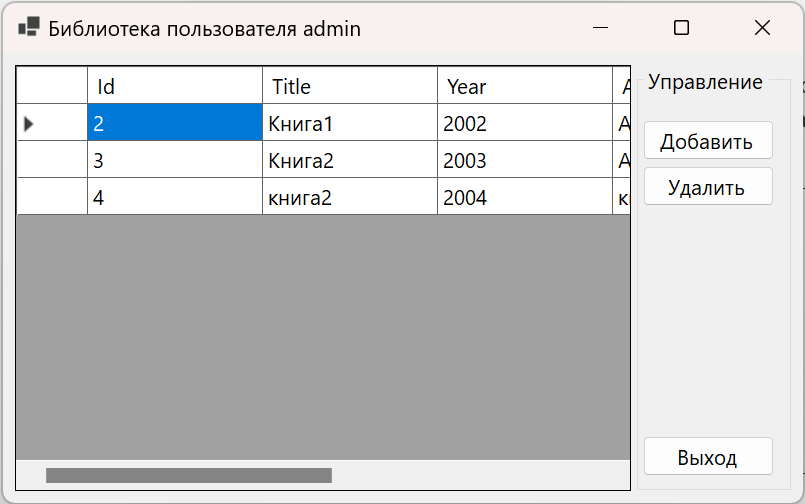


Рисунок 6 - Главная форма

При нажатии на кнопку "Добавить книгу" мы создаем новую форму для добавления книги и передаем в нее контекст базы данных и текущего пользователя. Если пользователь успешно добавил новую книгу, мы сохраняем ее в базе данных и обновляем таблицу с книгами на главной форме.

При нажатии на кнопку "Удалить книгу" мы удаляем выбранную книгу из списка книг пользователя и сохраняем изменения в базе данных. Затем обновляем таблицу с книгами на главной форме.

**Шаг 7. Создание формы для добавления новых книг**

Наконец, создадим форму для добавления новых книг. Добавим текстовые поля для ввода названия, автора и года выпуска книги, а также кнопку "Добавить".

|  |
| --- |
| public partial class BookForm : Form  {  private AppDbContext context;  private User user;  public Book Book { get; private set; }  public BookForm(AppDbContext context, User user)  {  InitializeComponent();  this.context = context;  this.user = user;  }  private void AddButton\_Click(object sender, EventArgs e)  {  var book = new Book  {  Title = TitleTextBox.Text,  Author = AuthorTextBox.Text,  Year = int.Parse(YearTextBox.Text),  UserId = user.Id  };  context.Books.Add(book);  context.SaveChanges();  Book = book;  DialogResult = DialogResult.OK;  }  } |

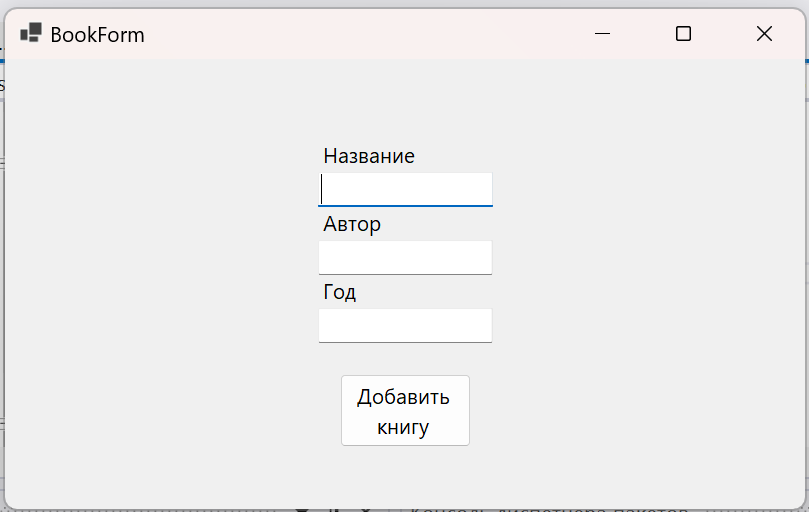


Рисунок 7 - Форма добавления книги

**Заключение**

В итоге вы создали приложение для управления списком книг, добавленных пользователями, с возможностью регистрации и авторизации. Мы использовали Entity Framework Core 7 для создания базы данных и работы с сущностями, Code First подход для создания таблиц на основе моделей, и Windows Forms для создания интерфейса пользователя.

**Варианты заданий**

1. Создание приложения для учета задач с возможностью добавления задач, их удаления и изменения статуса. Сущности: задача (название, описание, статус), пользователь (имя, email, пароль). Связи: один пользователь может иметь множество задач, каждая задача принадлежит одному пользователю.
2. Создание приложения для учета заказов с возможностью добавления заказов, их удаления и изменения статуса. Сущности: заказ (название, описание, статус), клиент (имя, email, телефон). Связи: один клиент может иметь множество заказов.
3. Создание приложения для учета студентов с возможностью добавления, удаления и изменения информации о студентах. Сущности: студент (имя, фамилия, дата рождения, группа), группа (название, курс, факультет), преподаватель (имя, email, пароль). Связи: каждая группа может содержать несколько студентов, каждый студент принадлежит одной группе.
4. Создание приложения для учета продаж с возможностью добавления продаж, их удаления и изменения статуса. Сущности: продажа (дата, клиент, товар, количество), клиент (имя, email, телефон). Связи: один клиент может совершать множество покупок.
5. Создание приложения для учета заявок на ремонт с возможностью добавления заявок, их удаления и изменения статуса. Сущности: заявка (название, описание, статус, дата создания, дата завершения), клиент (имя, email, телефон). Связи: один клиент может иметь множество заявок на ремонт.
6. Создание приложения для учета расходов с возможностью добавления расходов, их удаления и изменения статуса. Сущности: расход (название, описание, сумма, дата), пользователь (имя, email, пароль). Связи: один пользователь может иметь множество расходов.
7. Создание приложения для учета сотрудников с возможностью добавления, удаления и изменения информации о сотрудниках. Сущности: сотрудник (имя, фамилия, должность, email, телефон), отдел (название). Связи: один отдел может содержать множество сотрудников.
8. Создание приложения для учета продаж с возможностью добавления продаж, их удаления и изменения статуса. Сущности: продажа (дата, клиент, товар, количество), сотрудник (имя, фамилия, должность, email, телефон). Связи: один сотрудник может совершать множество продаж.
9. Создание приложения для учета встреч с возможностью добавления встреч, их удаления и изменения статуса. Сущности: встреча (название, описание, статус), пользователь (имя, email, пароль). Связи: один пользователь может иметь множество встреч, каждая встреча принадлежит одному пользователю. Функционал: добавление, удаление и изменение встреч, авторизация и регистрация пользователей, отображение списка встреч пользователя.
10. Создание приложения для учета событий с возможностью добавления, удаления и изменения событий. Сущности: событие (название, описание, дата начала, дата окончания), пользователь (имя, email, пароль). Связи: один пользователь может иметь множество событий.
11. Создание приложения для учета заметок с возможностью добавления, удаления и изменения заметок. Сущности: заметка (название, описание, дата создания), пользователь (имя, email, пароль). Связи: один пользователь может иметь множество заметок.
12. Создание приложения для учета счетов с возможностью добавления, удаления и изменения счетов. Сущности: счет (название, описание, баланс), пользователь (имя, email, пароль). Связи: один пользователь может иметь множество счетов.
13. Создание приложения для учета контрактов с возможностью добавления контрактов, их удаления и изменения статуса. Сущности: контракт (название, описание, статус, дата начала, дата окончания), компания (название, адрес, контактный email). Связи: каждый контракт заключается с определенной компанией, компания может иметь несколько контрактов.
14. Создание приложения для учета проектов по строительству с возможностью добавления проектов, их удаления и изменения статуса. Сущности: проект (название, описание, статус, адрес), инженер (имя, email, специализация). Связи: каждый проект имеет одного инженера, инженер может работать над несколькими проектами.
15. Создание приложения для учета проектов по строительству с возможностью добавления проектов, их удаления и изменения статуса. Сущности: проект (название, описание, статус, адрес), инженер (имя, email, специализация). Связи: каждый проект имеет одного инженера, инженер может работать над несколькими проектами.
16. Создание приложения для учета проектов по строительству с возможностью добавления проектов, их удаления и изменения статуса. Сущности: проект (название, описание, статус, адрес), инженер (имя, email, специализация). Связи: каждый проект имеет одного инженера, инженер может работать над несколькими проектами.
17. Создание приложения для учета проектов по строительству с возможностью добавления проектов, их удаления и изменения статуса. Сущности: проект (название, описание, статус, адрес), инженер (имя, email, специализация). Связи: каждый проект имеет одного инженера, инженер может работать над несколькими проектами.
18. Создание приложения для учета проектов по строительству с возможностью добавления проектов, их удаления и изменения статуса. Сущности: проект (название, описание, статус, адрес), инженер (имя, email, специализация). Связи: каждый проект имеет одного инженера, инженер может работать над несколькими проектами.