Реализация модулей

Информационная система "Радость бухгалтера"

1. Модуль учета сотрудников (Employee Management)

1.1. Описание модуля

Назначение: управление данными о сотрудниках организации.

Функции модуля:

- Создание, редактирование и удаление сотрудников
- Поиск сотрудников по ФИО
- Просмотр списка всех сотрудников
- Привязка сотрудников к должностям и отделам

1.2. Листинг модели Employee.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;

namespace PayrollSystem.Models
{
    public class Employee
    {
        public int EmployeeID { get; set; }
        public string LastName { get; set; }
        public string FirstName { get; set; }
        public string MiddleName { get; set; }
        public DateTime BirthDate { get; set; }
        public DateTime HireDate { get; set; }
        public int PositionID { get; set; }
        public int DepartmentID { get; set; }
        public decimal BaseSalary { get; set; }
        public virtual Position Position { get; set; }
        public virtual Department Department { get; set; }
        public virtual ICollection<Timesheet> Timesheets { get; set; }
        public virtual ICollection<Payroll> Payrolls { get; set; }

        [NotMapped]
        public string FullName => $"{LastName} {FirstName} {MiddleName}".Trim();

        [NotMapped]
        public string PositionName => Position?.PositionName ?? "";

        [NotMapped]
        public string DepartmentName => Department?.DepartmentName ?? "";
    }
}
```

1.3. Листинг страницы EmployeesPage.xaml.cs

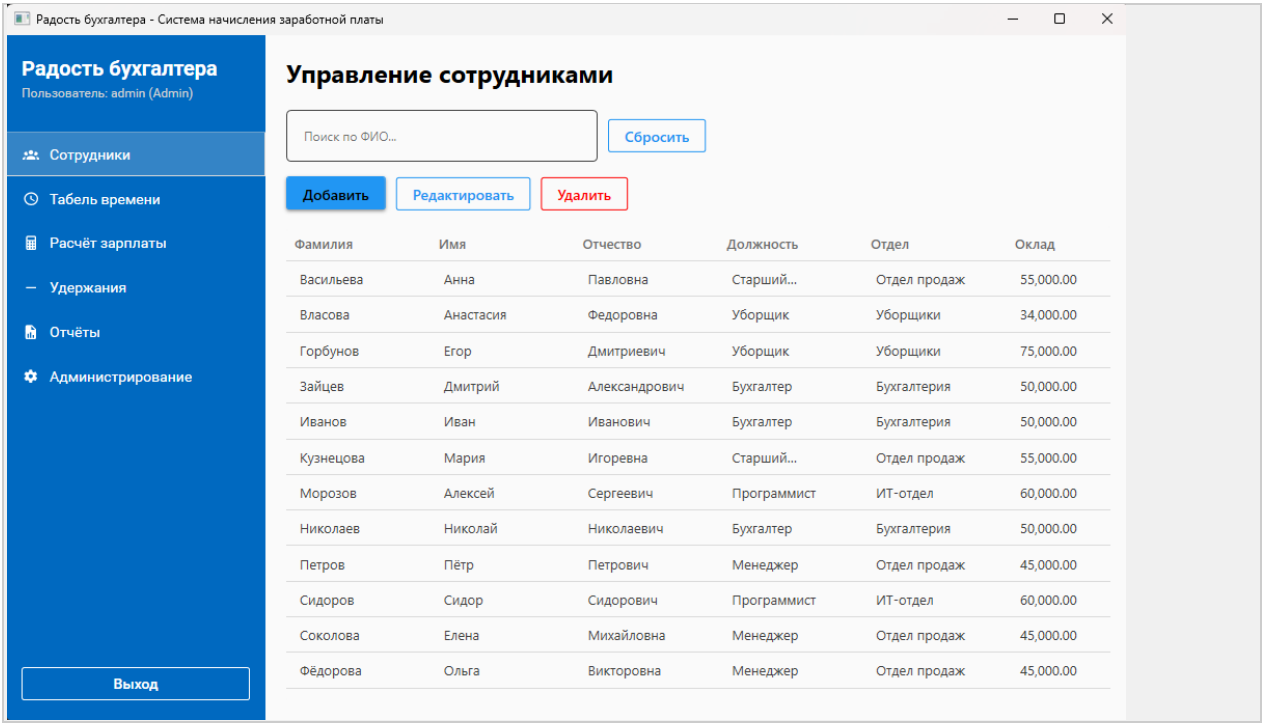
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using PayrollSystem.Models;
```

```
using PayrollSystem.Services;

namespace PayrollSystem.Views
{
    public partial class EmployeesPage : Page
    {
        private readonly DatabaseService _db;
        private List<Employee> _allEmployees;

        public EmployeesPage()
        {
            InitializeComponent();
            _db = new DatabaseService();
            LoadData();
        }

        private void LoadData()
        {
            try
            {
                _allEmployees = _db.GetAllEmployees();
                dgEmployees.ItemsSource = _allEmployees;
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка загрузки данных: {ex.Message}", "Ошибка",
                    MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }

        private void TxtSearch_TextChanged(object sender, TextChangedEventArgs e)
        {
            FilterEmployees();
        }

        private void FilterEmployees()
        {
            if (_allEmployees == null) return;

            string search = txtSearch.Text.Trim().ToLower();

            if (string.IsNullOrEmpty(search))
            {
                dgEmployees.ItemsSource = _allEmployees;
            }
            else
            {
                dgEmployees.ItemsSource = _allEmployees.Where(emp =>
                    emp.LastName.ToLower().Contains(search) ||
                    emp.FirstName.ToLower().Contains(search) ||
                    (emp.MiddleName != null && emp.MiddleName.ToLower().Contains(search))
                ).ToList();
            }
        }

        private void BtnResetSearch_Click(object sender, RoutedEventArgs e)
        {
            txtSearch.Text = "";
            dgEmployees.ItemsSource = _allEmployees;
        }

        private void BtnAdd_Click(object sender, RoutedEventArgs e)
        {
            var dialog = new EmployeeDialog();
            if (dialog.ShowDialog() == true)
            {
                LoadData();
            }
        }

        private void BtnEdit_Click(object sender, RoutedEventArgs e)
        {
            if (dgEmployees.SelectedItem is Employee emp)
            {
                var dialog = new EmployeeDialog(emp);
                if (dialog.ShowDialog() == true)
                {
```

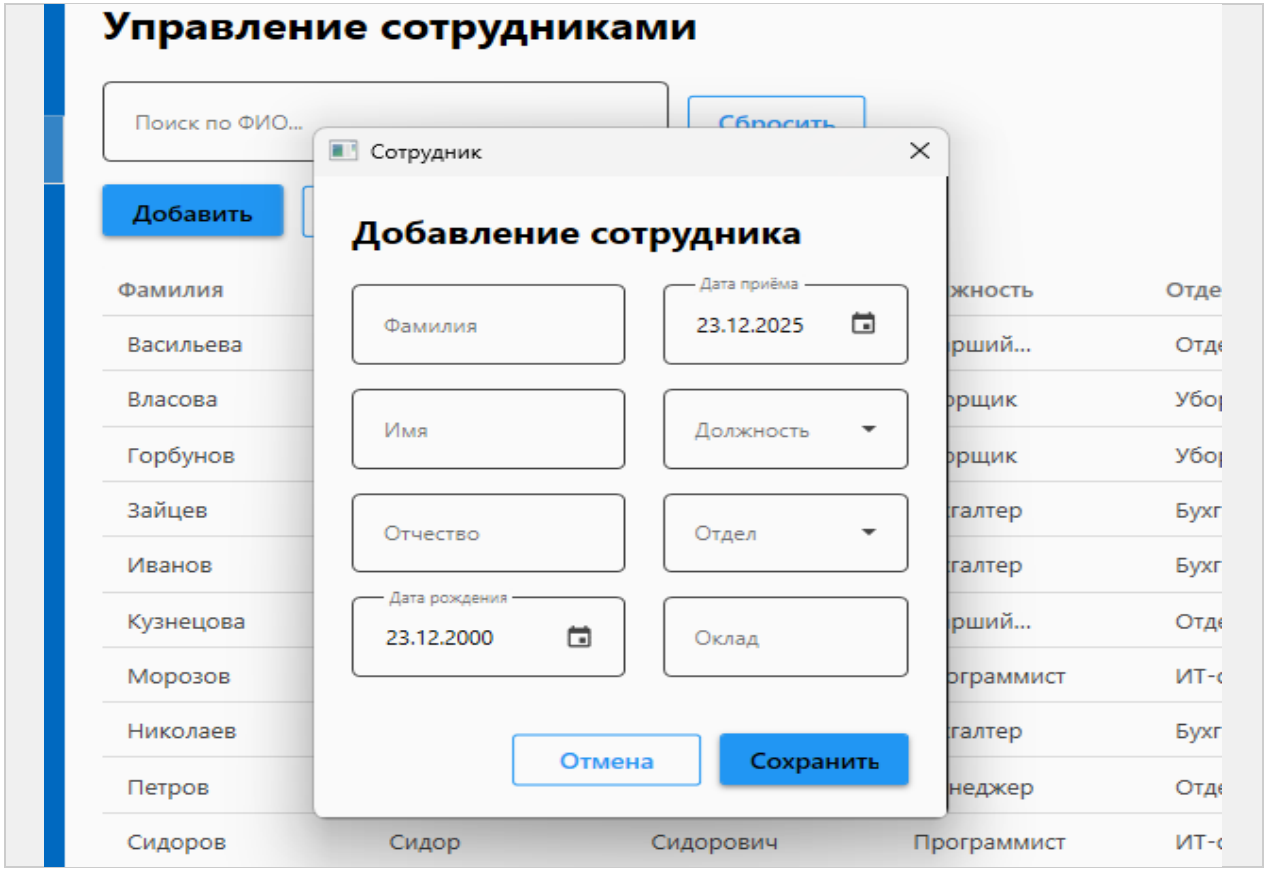
```
                    LoadData();
                }
            }
            else
            {
                MessageBox.Show("Выберите сотрудника для редактирования", "Внимание",
                    MessageBoxButton.OK, MessageBoxImage.Warning);
            }
        }

        private void BtnDelete_Click(object sender, RoutedEventArgs e)
        {
            if (dgEmployees.SelectedItem is Employee emp)
            {
                var result = MessageBox.Show(
                    $"Удалить сотрудника {emp.FullName}?",
                    "Подтверждение",
                    MessageBoxButton.YesNo,
                    MessageBoxImage.Question);

                if (result == MessageBoxResult.Yes)
                {
                    try
                    {
                        _db.DeleteEmployee(emp.EmployeeID);
                        LoadData();
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($"Ошибка удаления: {ex.Message}", "Ошибка",
                            MessageBoxButton.OK, MessageBoxImage.Error);
                    }
                }
            }
            else
            {
                MessageBox.Show("Выберите сотрудника для удаления", "Внимание",
                    MessageBoxButton.OK, MessageBoxImage.Warning);
            }
        }
    }
}
```

1.4. Скриншоты работы модуля

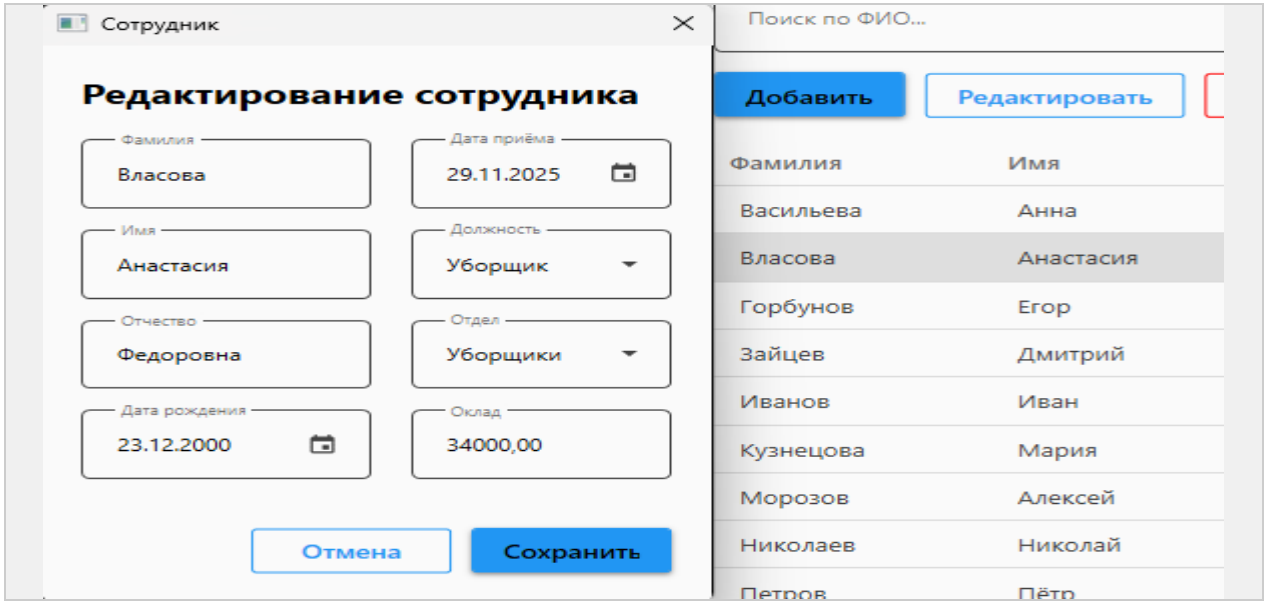
[СКРИНШОТ: Список сотрудников с функцией поиска]



[СКРИНШОТ: Диалоговое окно добавления сотрудника]



[СКРИНШОТ: Диалоговое окно редактирования сотрудника]



2. Модуль учета рабочего времени (Timesheet Management)

2.1. Описание модуля

Назначение: регистрация и учет отработанного времени сотрудников.

Функции модуля:

- Ввод табеля учета рабочего времени
- Редактирование и удаление записей табеля
- Просмотр истории отработанного времени
- Проверка корректности введенных данных

2.2. Листинг модели Timesheet.cs

```
using System;
using System.ComponentModel.DataAnnotations.Schema;

namespace PayrollSystem.Models
{
    public class Timesheet
    {
        public int TimesheetID { get; set; }
        public int EmployeeID { get; set; }
        public DateTime DateWorked { get; set; }
        public decimal Hours { get; set; }
        public virtual Employee Employee { get; set; }

        [NotMapped]
        public string EmployeeName => Employee != null ? $"{Employee.LastName} {Employee.FirstName}" : "";
    }
}
```

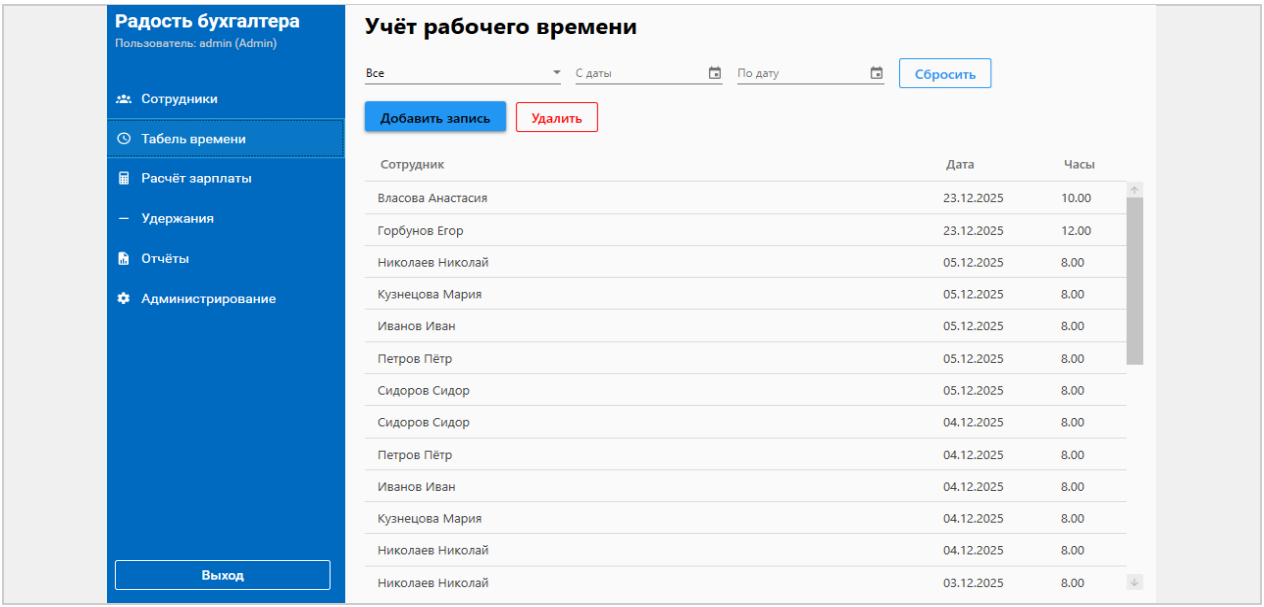
2.3. Листинг страницы TimesheetPage.xaml.cs

```
using System;
using System.Windows;
using System.Windows.Controls;
using PayrollSystem.Models;
using PayrollSystem.Services;

namespace PayrollSystem.Views
{
    public partial class TimesheetPage : Page
    {
        private readonly DatabaseService _db;

        public TimesheetPage()
        {
            InitializeComponent();
            _db = new DatabaseService();
            LoadEmployees();
            LoadData();
        }

        private void LoadEmployees()
        {
            var employees = _db.GetAllEmployees();
            employees.Insert(0, new Employee { EmployeeID = 0, FirstName = "Все", LastName = "" });
```

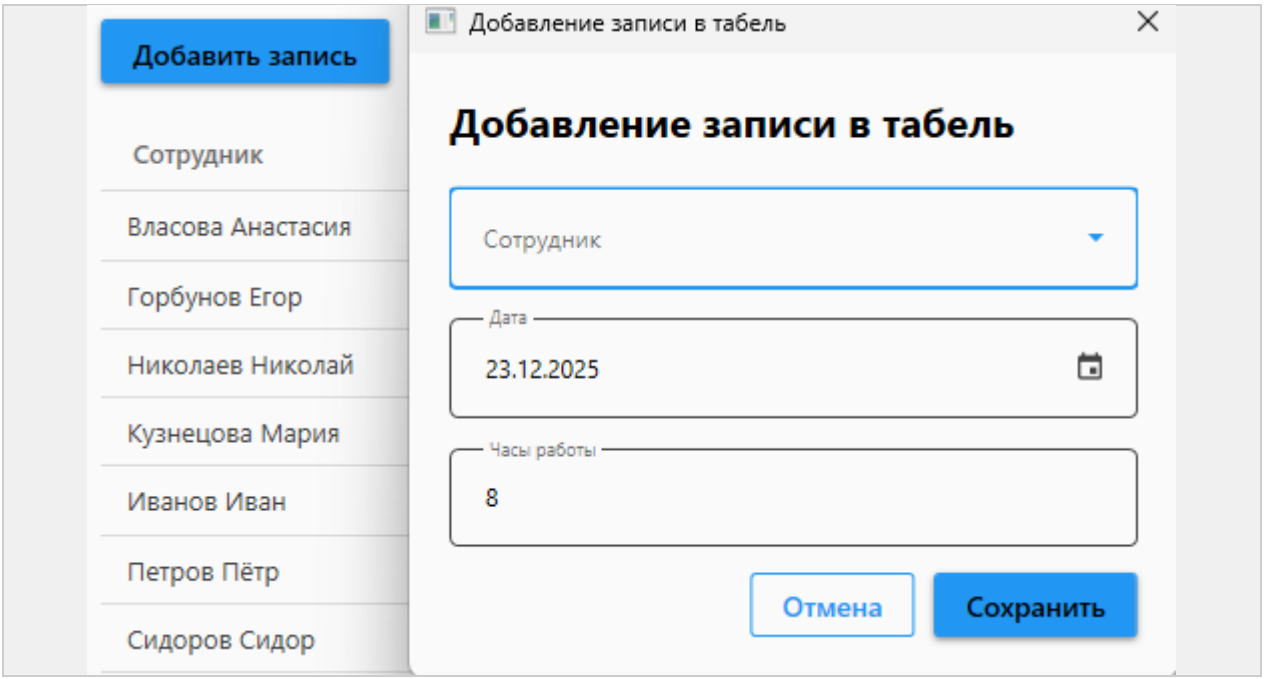
```
            cmbEmployee.ItemsSource = employees;
            cmbEmployee.SelectedIndex = 0;
        }

        private void LoadData()
        {
            try
            {
                int? empId = null;
                if (cmbEmployee.SelectedValue != null && (int)cmbEmployee.SelectedValue > 0)
                    empId = (int)cmbEmployee.SelectedValue;

                dgTimesheet.ItemsSource = _db.GetTimesheets(empId, dpStartDate.SelectedDate,
dpEndDate.SelectedDate);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка загрузки: {ex.Message}", "Ошибка",
                    MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }

        private void Filter_Changed(object sender, EventArgs e)
        {
            LoadData();
        }

        private void BtnReset_Click(object sender, RoutedEventArgs e)
        {
            cmbEmployee.SelectedIndex = 0;
            dpStartDate.SelectedDate = null;
            dpEndDate.SelectedDate = null;
            LoadData();
        }

        private void BtnAdd_Click(object sender, RoutedEventArgs e)
        {
            var dialog = new TimesheetDialog();
            if (dialog.ShowDialog() == true)
            {
                LoadData();
            }
        }

        private void BtnDelete_Click(object sender, RoutedEventArgs e)
        {
            if (dgTimesheet.SelectedItem is Timesheet ts)
            {
                var result = MessageBox.Show(
                    "Удалить эту запись?",
                    "Подтверждение",
                    MessageBoxButton.YesNo,
                    MessageBoxImage.Question);

                if (result == MessageBoxResult.Yes)
                {
                    try
                    {
                        _db.DeleteTimesheet(ts.TimesheetID);
                        LoadData();
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($"Ошибка удаления: {ex.Message}", "Ошибка",
                            MessageBoxButton.OK, MessageBoxImage.Error);
                    }
                }
            }
            else
            {
                MessageBox.Show("Выберите запись для удаления", "Внимание",
                    MessageBoxButton.OK, MessageBoxImage.Warning);
            }
        }
    }
}
```

2.4. Скриншоты работы модуля

[СКРИНШОТ: Список записей табеля учета времени]



[СКРИНШОТ: Диалоговое окно добавления записи табеля]



3. Модуль расчета заработной платы (Payroll Calculation)

3.1. Описание модуля

Назначение: автоматический расчет заработной платы с учетом окладов, премий и удержаний.

Функции модуля:

- Расчет зарплаты за выбранный период
- Индивидуальный расчет с учетом бонусов и штрафов
- Автоматическое применение удержаний
- Формирование итоговых сумм для выплат
- Просмотр истории начислений

3.2. Листинг модели Payroll.cs

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;

namespace PayrollSystem.Models
{
    public class Payroll
    {
        public int PayrollID { get; set; }
        public int EmployeeID { get; set; }
        public string Period { get; set; }
        public decimal BaseSalary { get; set; }
        public decimal WorkedHours { get; set; }
        public decimal Bonus { get; set; }
        public decimal Penalty { get; set; }
        public decimal NetSalary { get; set; }
        public virtual Employee Employee { get; set; }
        public virtual ICollection<PayrollDeduction> PayrollDeductions { get; set; }

        [NotMapped]
        public string EmployeeName => Employee != null ? $"{Employee.LastName} {Employee.FirstName}" : "";

        [NotMapped]
        public decimal TotalDeductions => PayrollDeductions?.Sum(pd => pd.Amount) ?? 0;
    }
}
```

3.3. Листинг страницы PayrollPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using PayrollSystem.Models;
using PayrollSystem.Services;

namespace PayrollSystem.Views
{
    public partial class PayrollPage : Page
    {
        private readonly DatabaseService _db;
        private List<Payroll> _allPayrolls;

        public PayrollPage()
        {
            InitializeComponent();
            _db = new DatabaseService();
            InitializeComboBoxes();
```

```
        }

        private void InitializeComboBoxes()
        {
            cmbMonth.Items.Add(new { Value = "01", Name = "Январь" });
            cmbMonth.Items.Add(new { Value = "02", Name = "Февраль" });
            cmbMonth.Items.Add(new { Value = "03", Name = "Март" });
            cmbMonth.Items.Add(new { Value = "04", Name = "Апрель" });
            cmbMonth.Items.Add(new { Value = "05", Name = "Май" });
            cmbMonth.Items.Add(new { Value = "06", Name = "Июнь" });
            cmbMonth.Items.Add(new { Value = "07", Name = "Июль" });
            cmbMonth.Items.Add(new { Value = "08", Name = "Август" });
            cmbMonth.Items.Add(new { Value = "09", Name = "Сентябрь" });
            cmbMonth.Items.Add(new { Value = "10", Name = "Октябрь" });
            cmbMonth.Items.Add(new { Value = "11", Name = "Ноябрь" });
            cmbMonth.Items.Add(new { Value = "12", Name = "Декабрь" });
            cmbMonth.DisplayMemberPath = "Name";
            cmbMonth.SelectedValuePath = "Value";
            cmbMonth.SelectedIndex = DateTime.Now.Month - 1;

            for (int year = 2020; year <= DateTime.Now.Year + 1; year++)
            {
                cmbYear.Items.Add(year);
            }
            cmbYear.SelectedItem = DateTime.Now.Year;
        }

        private string GetSelectedPeriod()
        {
            if (cmbYear.SelectedItem == null || cmbMonth.SelectedValue == null)
                return null;
            return $"{cmbYear.SelectedItem}-{cmbMonth.SelectedValue}";
        }

        private void BtnCalculate_Click(object sender, RoutedEventArgs e)
        {
            string period = GetSelectedPeriod();
            if (period == null)
            {
                MessageBox.Show("Выберите период", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
                return;
            }

            try
            {
                _db.CalculatePayrollForAll(period);
                MessageBox.Show("Расчёт выполнен!", "Готово", MessageBoxButton.OK,
MessageBoxImage.Information);
                LoadData(period);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка расчёта: {ex.Message}", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void BtnShow_Click(object sender, RoutedEventArgs e)
        {
            string period = GetSelectedPeriod();
            if (period == null)
            {
                MessageBox.Show("Выберите период", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
                return;
            }
            LoadData(period);
        }

        private void LoadData(string period)
        {
            try
            {
                _allPayrolls = _db.GetPayrolls(period);
                ApplyFilter();
                txtCalculationDetails.Text = "Выберите сотрудника для просмотра расчёта...";
```

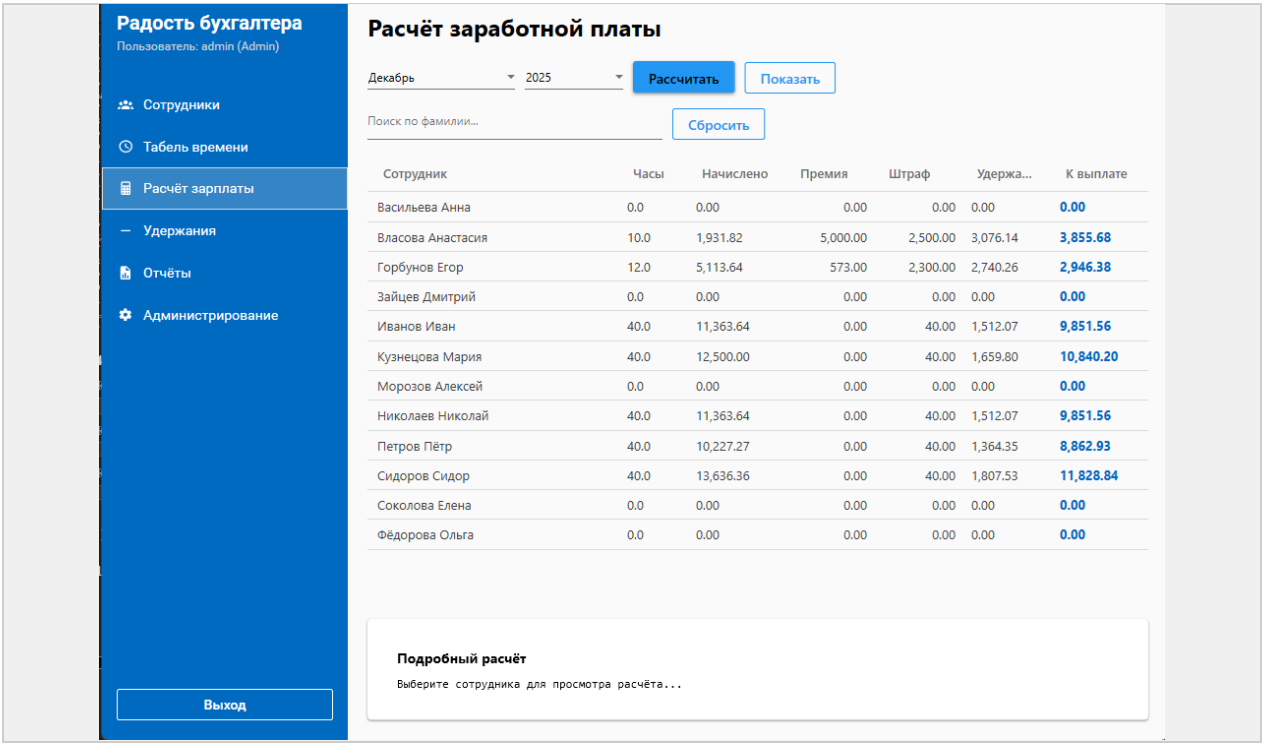
```
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка загрузки: {ex.Message}", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void TxtSearch_TextChanged(object sender, TextChangedEventArgs e)
        {
            ApplyFilter();
        }

        private void BtnResetSearch_Click(object sender, RoutedEventArgs e)
        {
            txtSearch.Text = "";
            ApplyFilter();
        }

        private void ApplyFilter()
        {
            if (_allPayrolls == null) return;

            string search = txtSearch.Text.Trim().ToLower();

            if (string.IsNullOrEmpty(search))
            {
                dgPayroll.ItemsSource = _allPayrolls;
            }
            else
            {
                dgPayroll.ItemsSource = _allPayrolls.Where(p =>
                    p.EmployeeName.ToLower().Contains(search)
                ).ToList();
            }
        }

        private void DgPayroll_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            if (dgPayroll.SelectedItem is Payroll payroll)
            {
                ShowCalculationDetails(payroll);
            }
        }

        private void ShowCalculationDetails(Payroll payroll)
        {
            try
            {
                string period = GetSelectedPeriod();
                if (period == null) return;

                decimal normHours = _db.GetNormHoursByPeriod(period);
                decimal employeeSalary = _db.GetEmployeeSalary(payroll.EmployeeID);
                decimal hourlyRate = normHours > 0 ? employeeSalary / normHours : 0;
                decimal workedHours = payroll.WorkedHours;

                var sb = new StringBuilder();
                sb.AppendLine($"═══ Сотрудник: {payroll.EmployeeName} ═══");
                sb.AppendLine();
                sb.AppendLine($"► Часы: {workedHours:N1} (отработано) / {normHours:N0} (норма по произв.
календарю)");
                sb.AppendLine($"► Оклад: {employeeSalary:N2} руб.");
                sb.AppendLine($"► Часовая ставка: {employeeSalary:N2} / {normHours:N0} = {hourlyRate:N2}
руб./час");
                sb.AppendLine();

                decimal baseSalaryCalc;
                if (workedHours <= normHours)
                {
                    baseSalaryCalc = hourlyRate * workedHours;
                    sb.AppendLine("► Расчёт (часов ≤ нормы):");
                    sb.AppendLine($"   Начислено = {hourlyRate:N2} × {workedHours:N1} =
{baseSalaryCalc:N2} руб.");
                }
                else
                {
```

```
                    decimal overtime = workedHours - normHours;
                    decimal overtimePay = overtime * hourlyRate * 1.5m;
                    baseSalaryCalc = employeeSalary + overtimePay;
                    sb.AppendLine("► Расчёт (есть переработка):");
                    sb.AppendLine($"   Основная часть: {employeeSalary:N2} руб. (полный оклад)");
                    sb.AppendLine($"   Переработка: {overtime:N1} час × {hourlyRate:N2} × 1.5 =
{overtimePay:N2} руб.");
                    sb.AppendLine($"   Начислено = {employeeSalary:N2} + {overtimePay:N2} =
{baseSalaryCalc:N2} руб.");
                }

                sb.AppendLine();
                sb.AppendLine($"► Премия: +{payroll.Bonus:N2} руб.");
                sb.AppendLine($"► Штраф: -{payroll.Penalty:N2} руб.");

                decimal taxBase = baseSalaryCalc + payroll.Bonus - payroll.Penalty;
                if (taxBase < 0) taxBase = 0;
                decimal tax = taxBase * 0.13m;

                sb.AppendLine();
                sb.AppendLine("► База для налога:");
                sb.AppendLine($"   {baseSalaryCalc:N2} + {payroll.Bonus:N2} - {payroll.Penalty:N2} =
{taxBase:N2} руб.");
                sb.AppendLine();
                sb.AppendLine("► Удержания:");
                sb.AppendLine($"   НДФЛ 13%: {taxBase:N2} × 13% = {tax:N2} руб.");
                if (payroll.Penalty > 0)
                {
                    sb.AppendLine($"   Штраф: {payroll.Penalty:N2} руб.");
                }
                sb.AppendLine($"   Итого удержано: {tax + payroll.Penalty:N2} руб.");

                sb.AppendLine();
                sb.AppendLine("═══════════════════════════════════════════");
                decimal netCalc = taxBase - tax;
                sb.AppendLine($"► Формула: (Начислено + Премия - Штраф) - 13%");
                sb.AppendLine($"► К ВЫПЛАТЕ: {taxBase:N2} - {tax:N2} = {netCalc:N2} руб.");
                sb.AppendLine("═══════════════════════════════════════════");

                txtCalculationDetails.Text = sb.ToString();
            }
            catch (Exception ex)
            {
                txtCalculationDetails.Text = $"Ошибка получения данных: {ex.Message}";
            }
        }

        private void DgPayroll_CellEditEnding(object sender, DataGridCellEditEndingEventArgs e)
        {
            if (e.EditAction == DataGridEditAction.Cancel) return;

            if (e.Row.Item is Payroll payroll)
            {
                var textBox = e.EditingElement as TextBox;
                if (textBox == null) return;

                string columnHeader = e.Column.Header.ToString();
                decimal newValue;

                if (!decimal.TryParse(textBox.Text, out newValue))
                {
                    MessageBox.Show("Введите корректное число", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Warning);
                    e.Cancel = true;
                    return;
                }

                if (newValue < 0)
                {
                    MessageBox.Show("Значение не может быть отрицательным", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Warning);
                    e.Cancel = true;
                    return;
                }

                try
                {
```

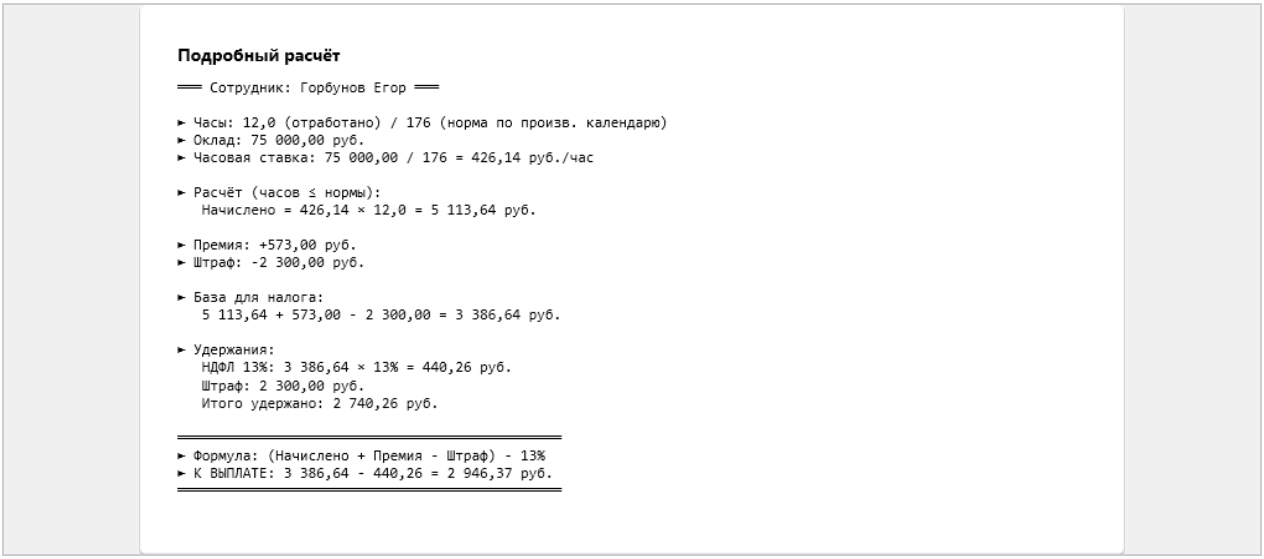
```
                    decimal bonus = payroll.Bonus;
                    decimal penalty = payroll.Penalty;

                    if (columnHeader == "Премия")
                        bonus = newValue;
                    else if (columnHeader == "Штраф")
                        penalty = newValue;

                    _db.UpdatePayrollBonusAndPenalty(payroll.PayrollID, bonus, penalty);

                    Dispatcher.BeginInvoke(new Action(() =>
                    {
                        string period = GetSelectedPeriod();
                        LoadData(period);

                        var updated = _allPayrolls?.FirstOrDefault(p => p.PayrollID == payroll.PayrollID);
                        if (updated != null)
                        {
                            dgPayroll.SelectedItem = updated;
                            ShowCalculationDetails(updated);
                        }
                    }), System.Windows.Threading.DispatcherPriority.Background);
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Ошибка сохранения: {ex.Message}", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
                    e.Cancel = true;
                }
            }
        }
    }
}
```

3.4. Скриншоты работы модуля

[СКРИНШОТ: Список начислений заработной платы]



[СКРИНШОТ: Результат расчета с детализацией]



Подробный расчёт

```
═══ Сотрудник: Горбунов Егор ═══

► Часы: 12,0 (отработано) / 176 (норма по произв. календарю)
► Оклад: 75 000,00 руб.
► Часовая ставка: 75 000,00 / 176 = 426,14 руб./час

► Расчёт (часов ≤ нормы):
   Начислено = 426,14 × 12,0 = 5 113,64 руб.

► Премия: +573,00 руб.
► Штраф: -2 300,00 руб.

► База для налога:
   5 113,64 + 573,00 - 2 300,00 = 3 386,64 руб.

► Удержания:
   НДФЛ 13%: 3 386,64 × 13% = 440,26 руб.
   Штраф: 2 300,00 руб.
   Итого удержано: 2 740,26 руб.

═══════════════════════════════════════
► Формула: (Начислено + Премия - Штраф) - 13%
► К ВЫПЛАТЕ: 3 386,64 - 440,26 = 2 946,37 руб.
═══════════════════════════════════════
```

[СКРИНШОТ: Добавление бонусов/штрафов сотрудникам]

Сотрудник	Часы	Начислено	Премия	Штраф	Удержа...	К выплате
Власова Анастасия	10.0	1,931.82	5,000.00	2,500.00	3,076.14	**3,855.68**
Горбунов Егор	12.0	5,113.64	573.00	1300	2,740.26	**2,946.37**
Зайцев Дмитрий	0.0	0.00	0.00	0.00	0.00	**0.00**

4. Модуль учета удержаний (Deductions Management)

4.1. Описание модуля

Назначение: ведение информации об удержаниях и их привязка к начислениям.

Функции модуля:

- Добавление, редактирование и удаление удержаний
- Хранение справочника видов удержаний
- Привязка удержаний к конкретным расчетам зарплаты
- Просмотр истории удержаний

4.2. Листинг модели Deduction.cs

```
using System.Collections.Generic;

namespace PayrollSystem.Models
{
    public class Deduction
    {
        public int DeductionID { get; set; }
        public string DeductionName { get; set; }
        public virtual ICollection<PayrollDeduction> PayrollDeductions { get; set; }
    }
}
```

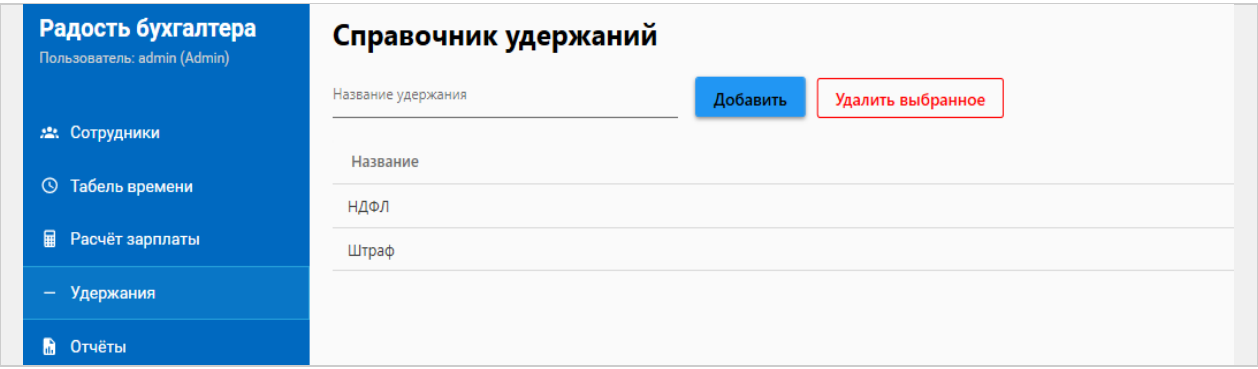
4.3. Листинг страницы DeductionsPage.xaml.cs

```
using System;
using System.Windows;
using System.Windows.Controls;
using PayrollSystem.Models;
using PayrollSystem.Services;

namespace PayrollSystem.Views
{
    public partial class DeductionsPage : Page
    {
        private readonly DatabaseService _db;

        public DeductionsPage()
        {
            InitializeComponent();
            _db = new DatabaseService();
            LoadData();
        }

        private void LoadData()
        {
            try
            {
                dgDeductions.ItemsSource = _db.GetAllDeductions();
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка загрузки: {ex.Message}", "Ошибка",
                    MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }

        private void BtnAdd_Click(object sender, RoutedEventArgs e)
        {
            string name = txtDeductionName.Text.Trim();
            if (string.IsNullOrEmpty(name))
            {
                MessageBox.Show("Введите название удержания", "Внимание",
                    MessageBoxButton.OK, MessageBoxImage.Warning);
                return;
            }

            try
            {
                _db.AddDeduction(new Deduction { DeductionName = name });
                txtDeductionName.Text = "";
                LoadData();
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка добавления: {ex.Message}", "Ошибка",
                    MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }

        private void BtnDelete_Click(object sender, RoutedEventArgs e)
        {
            if (dgDeductions.SelectedItem is Deduction d)
            {
                if (MessageBox.Show($"Удалить удержание \"{d.DeductionName}\"?", "Подтверждение",
                    MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)
                {
                    try
                    {
                        _db.DeleteDeduction(d.DeductionID);
                        LoadData();
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($"Ошибка удаления: {ex.Message}", "Ошибка",
                            MessageBoxButton.OK, MessageBoxImage.Error);
                    }
                }
            }
            else
```

```
            {
                MessageBox.Show("Выберите удержание для удаления", "Внимание",
                    MessageBoxButton.OK, MessageBoxImage.Warning);
            }
        }
    }
}
```

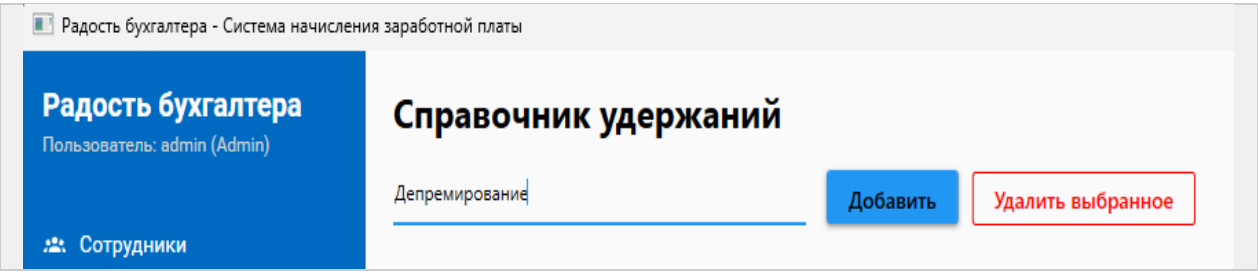
4.4. Скриншоты работы модуля

[СКРИНШОТ: Список видов удержаний]



[СКРИНШОТ: Диалоговое окно добавления удержания]

Возможность добавить удержание имеется, но функционал его пока не реализован.



5. Модуль формирования отчетов (Reporting)

5.1. Описание модуля

Назначение: генерация отчетов и расчетных листков для сотрудников и руководства.

Функции модуля:

- Формирование расчетных листков в PDF
- Создание отчетов по фонду оплаты труда в Excel
- Выбор периода и сотрудника для отчета
- Экспорт данных в различные форматы

5.2. Листинг страницы ReportsPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using Microsoft.Win32;
```

```
using PayrollSystem.Models;
using PayrollSystem.Services;

namespace PayrollSystem.Views
{
    public partial class ReportsPage : Page
    {
        private readonly DatabaseService _db;
        private List<Payroll> _currentPayrolls;

        public ReportsPage()
        {
            InitializeComponent();
            _db = new DatabaseService();
            InitializeComboBoxes();
            LoadStatistics();
        }

        private void InitializeComboBoxes()
        {
            cmbReportMonth.Items.Add(new { Value = "01", Name = "Январь" });
            cmbReportMonth.Items.Add(new { Value = "02", Name = "Февраль" });
            cmbReportMonth.Items.Add(new { Value = "03", Name = "Март" });
            cmbReportMonth.Items.Add(new { Value = "04", Name = "Апрель" });
            cmbReportMonth.Items.Add(new { Value = "05", Name = "Май" });
            cmbReportMonth.Items.Add(new { Value = "06", Name = "Июнь" });
            cmbReportMonth.Items.Add(new { Value = "07", Name = "Июль" });
            cmbReportMonth.Items.Add(new { Value = "08", Name = "Август" });
            cmbReportMonth.Items.Add(new { Value = "09", Name = "Сентябрь" });
            cmbReportMonth.Items.Add(new { Value = "10", Name = "Октябрь" });
            cmbReportMonth.Items.Add(new { Value = "11", Name = "Ноябрь" });
            cmbReportMonth.Items.Add(new { Value = "12", Name = "Декабрь" });
            cmbReportMonth.DisplayMemberPath = "Name";
            cmbReportMonth.SelectedValuePath = "Value";
            cmbReportMonth.SelectedIndex = DateTime.Now.Month - 1;

            for (int year = 2020; year <= DateTime.Now.Year + 1; year++)
            {
                cmbReportYear.Items.Add(year);
            }
            cmbReportYear.SelectedItem = DateTime.Now.Year;
        }

        private string GetSelectedPeriod()
        {
            if (cmbReportYear.SelectedItem == null || cmbReportMonth.SelectedValue == null)
                return null;
            return $"{cmbReportYear.SelectedItem}-{cmbReportMonth.SelectedValue}";
        }

        private void LoadStatistics()
        {
            try
            {
                txtEmployeeCount.Text = _db.GetEmployeeCount().ToString();
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка: {ex.Message}", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void BtnGenerateReport_Click(object sender, RoutedEventArgs e)
        {
            string period = GetSelectedPeriod();
            if (period == null)
            {
                MessageBox.Show("Выберите период", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
                return;
            }

            try
            {
                _currentPayrolls = _db.GetPayrolls(period);
                dgReport.ItemsSource = _currentPayrolls;
```

```
                decimal total = _currentPayrolls.Sum(p => p.NetSalary);
                txtReportTotal.Text = $"{total:N2} руб.";
                txtTotalPayroll.Text = $"{total:N2} руб.";
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка: {ex.Message}", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void BtnExportPdf_Click(object sender, RoutedEventArgs e)
        {
            if (_currentPayrolls == null || !_currentPayrolls.Any())
            {
                MessageBox.Show("Сначала сформируйте отчёт", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
                return;
            }

            var dialog = new SelectEmployeeForReportDialog(_currentPayrolls);
            if (dialog.ShowDialog() == true && dialog.SelectedPayroll != null)
            {
                var saveDialog = new SaveFileDialog
                {
                    Filter = "PDF файлы (*.pdf)|*.pdf",
                    FileName = $"Расчётный_лист_{dialog.SelectedPayroll.EmployeeName.Replace(" ",
"_")}_{dialog.SelectedPayroll.Period}.pdf"
                };

                if (saveDialog.ShowDialog() == true)
                {
                    try
                    {
                        var deductions = _db.GetPayrollDeductions(dialog.SelectedPayroll.PayrollID);
                        ExportService.ExportPayslipToPdf(dialog.SelectedPayroll, deductions,
saveDialog.FileName);
                        MessageBox.Show("Расчётный лист сохранён!", "Готово", MessageBoxButton.OK,
MessageBoxImage.Information);
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($"Ошибка экспорта: {ex.Message}", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
                    }
                }
            }
        }

        private void BtnExportExcel_Click(object sender, RoutedEventArgs e)
        {
            if (_currentPayrolls == null || !_currentPayrolls.Any())
            {
                MessageBox.Show("Сначала сформируйте отчёт", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
                return;
            }

            var saveDialog = new SaveFileDialog
            {
                Filter = "Excel файлы (*.xlsx)|*.xlsx",
                FileName = $"Зарплатная_ведомость_{GetSelectedPeriod()}.xlsx"
            };

            if (saveDialog.ShowDialog() == true)
            {
                try
                {
                    ExportService.ExportPayrollToExcel(_currentPayrolls, GetSelectedPeriod(),
saveDialog.FileName);
                    MessageBox.Show("Ведомость сохранена!", "Готово", MessageBoxButton.OK,
MessageBoxImage.Information);
                }
                catch (Exception ex)
                {
```

```
                    MessageBox.Show($"Ошибка экспорта: {ex.Message}", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
                }
            }
        }
    }
}
```

5.3. Листинг сервиса ExportService.cs

```
using System.Collections.Generic;
using System.IO;
using iTextSharp.text;
using iTextSharp.text.pdf;
using OfficeOpenXml;
using OfficeOpenXml.Style;
using PayrollSystem.Models;

namespace PayrollSystem.Services
{
    public static class ExportService
    {
        public static void ExportPayslipToPdf(Payroll payroll, List<PayrollDeduction> deductions, string
filePath)
        {
            var doc = new Document(PageSize.A4, 50, 50, 50, 50);
            PdfWriter.GetInstance(doc, new FileStream(filePath, FileMode.Create));
            doc.Open();

            string fontPath =
Path.Combine(System.Environment.GetFolderPath(System.Environment.SpecialFolder.Fonts), "arial.ttf");
            BaseFont baseFont = BaseFont.CreateFont(fontPath, BaseFont.IDENTITY_H, BaseFont.EMBEDDED);
            var titleFont = new Font(baseFont, 18, Font.BOLD);
            var headerFont = new Font(baseFont, 12, Font.BOLD);
            var normalFont = new Font(baseFont, 11, Font.NORMAL);
            var smallFont = new Font(baseFont, 10, Font.NORMAL);

            var title = new Paragraph("РАСЧЁТНЫЙ ЛИСТОК", titleFont)
            {
                Alignment = Element.ALIGN_CENTER
            };
            doc.Add(title);
            doc.Add(new Paragraph("\n"));

            var infoTable = new PdfPTable(2)
            {
                WidthPercentage = 100
            };
            infoTable.SetWidths(new float[] { 1, 2 });

            AddCell(infoTable, "Сотрудник:", headerFont);
            AddCell(infoTable, payroll.EmployeeName, normalFont);
            AddCell(infoTable, "Период:", headerFont);
            AddCell(infoTable, payroll.Period, normalFont);
            AddCell(infoTable, "Отработано часов:", headerFont);
            AddCell(infoTable, payroll.WorkedHours.ToString("N1"), normalFont);

            doc.Add(infoTable);
            doc.Add(new Paragraph("\n"));

            var accrualHeader = new Paragraph("НАЧИСЛЕНИЯ", headerFont);
            doc.Add(accrualHeader);

            var accrualTable = new PdfPTable(2)
            {
                WidthPercentage = 100
            };
            accrualTable.SetWidths(new float[] { 3, 1 });

            AddCell(accrualTable, "За отработанное время:", normalFont);
```

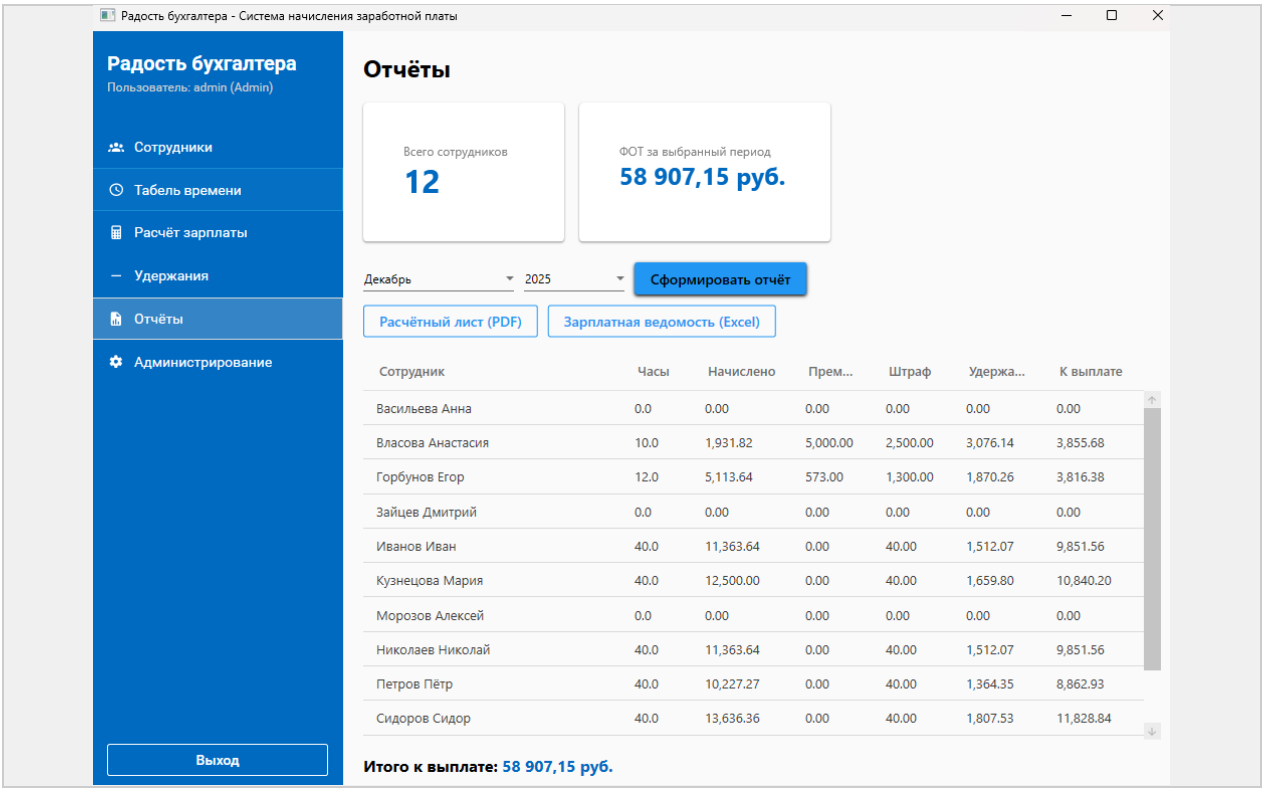
```
            AddCell(accrualTable, payroll.BaseSalary.ToString("N2") + " руб.", normalFont,
Element.ALIGN_RIGHT);
            AddCell(accrualTable, "Премия:", normalFont);
            AddCell(accrualTable, payroll.Bonus.ToString("N2") + " руб.", normalFont,
Element.ALIGN_RIGHT);

            decimal totalAccrual = payroll.BaseSalary + payroll.Bonus;
            var totalAccrualLabel = new PdfPCell(new Phrase("ИТОГО начислено:", headerFont))
            {
                Border = Rectangle.TOP_BORDER,
                PaddingTop = 5
            };
            accrualTable.AddCell(totalAccrualLabel);

            var totalAccrualValue = new PdfPCell(new Phrase(totalAccrual.ToString("N2") + " руб.",
headerFont))
            {
                Border = Rectangle.TOP_BORDER,
                HorizontalAlignment = Element.ALIGN_RIGHT,
                PaddingTop = 5
            };
            accrualTable.AddCell(totalAccrualValue);

            doc.Add(accrualTable);
            doc.Add(new Paragraph("\n"));

            var deductionHeader = new Paragraph("УДЕРЖАНИЯ", headerFont);
            doc.Add(deductionHeader);

            var deductionTable = new PdfPTable(2)
            {
                WidthPercentage = 100
            };
            deductionTable.SetWidths(new float[] { 3, 1 });

            decimal totalDeductions = 0;
            foreach (var d in deductions)
            {
                AddCell(deductionTable, d.DeductionName + ":", normalFont);
                AddCell(deductionTable, d.Amount.ToString("N2") + " руб.", normalFont,
Element.ALIGN_RIGHT);
                totalDeductions += d.Amount;
            }

            var totalDeductionLabel = new PdfPCell(new Phrase("ИТОГО удержано:", headerFont))
            {
                Border = Rectangle.TOP_BORDER,
                PaddingTop = 5
            };
            deductionTable.AddCell(totalDeductionLabel);

            var totalDeductionValue = new PdfPCell(new Phrase(totalDeductions.ToString("N2") + " руб.",
headerFont))
            {
                Border = Rectangle.TOP_BORDER,
                HorizontalAlignment = Element.ALIGN_RIGHT,
                PaddingTop = 5
            };
            deductionTable.AddCell(totalDeductionValue);

            doc.Add(deductionTable);
            doc.Add(new Paragraph("\n"));

            var totalTable = new PdfPTable(2)
            {
                WidthPercentage = 100
            };
            totalTable.SetWidths(new float[] { 3, 1 });

            var totalFont = new Font(baseFont, 14, Font.BOLD);

            var toPayLabel = new PdfPCell(new Phrase("К ВЫПЛАТЕ:", totalFont))
            {
                Border = Rectangle.BOX,
                Padding = 10,
                BackgroundColor = new BaseColor(230, 230, 230)
            };
```

```
            totalTable.AddCell(toPayLabel);

            var toPayValue = new PdfPCell(new Phrase(payroll.NetSalary.ToString("N2") + " руб.",
totalFont))
            {
                Border = Rectangle.BOX,
                Padding = 10,
                HorizontalAlignment = Element.ALIGN_RIGHT,
                BackgroundColor = new BaseColor(230, 230, 230)
            };
            totalTable.AddCell(toPayValue);

            doc.Add(totalTable);
            doc.Add(new Paragraph("\n\n"));

            doc.Add(new Paragraph("Бухгалтер: _____ / _____", smallFont));
            doc.Add(new Paragraph("\n"));
            doc.Add(new Paragraph("Дата: _____", smallFont));

            doc.Close();
        }

        private static void AddCell(PdfPTable table, string text, Font font, int alignment =
Element.ALIGN_LEFT)
        {
            var cell = new PdfPCell(new Phrase(text, font))
            {
                Border = Rectangle.NO_BORDER,
                HorizontalAlignment = alignment,
                Padding = 3
            };
            table.AddCell(cell);
        }

        public static void ExportPayrollToExcel(List<Payroll> payrolls, string period, string filePath)
        {
            using (var package = new ExcelPackage())
            {
                var ws = package.Workbook.Worksheets.Add("Ведомость");

                ws.Cells["A1"].Value = "ЗАРПЛАТНАЯ ВЕДОМОСТЬ";
                ws.Cells["A1:H1"].Merge = true;
                ws.Cells["A1"].Style.Font.Size = 16;
                ws.Cells["A1"].Style.Font.Bold = true;
                ws.Cells["A1"].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

                ws.Cells["A2"].Value = "Период: " + period;
                ws.Cells["A2:H2"].Merge = true;
                ws.Cells["A2"].Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

                int row = 4;
                ws.Cells[row, 1].Value = "№";
                ws.Cells[row, 2].Value = "Сотрудник";
                ws.Cells[row, 3].Value = "Часы";
                ws.Cells[row, 4].Value = "Начислено";
                ws.Cells[row, 5].Value = "Премия";
                ws.Cells[row, 6].Value = "Штраф";
                ws.Cells[row, 7].Value = "Удержания";
                ws.Cells[row, 8].Value = "К выплате";

                using (var range = ws.Cells[row, 1, row, 8])
                {
                    range.Style.Font.Bold = true;
                    range.Style.Fill.PatternType = ExcelFillStyle.Solid;
                    range.Style.Fill.BackgroundColor.SetColor(System.Drawing.Color.LightGray);
                    range.Style.Border.Bottom.Style = ExcelBorderStyle.Thin;
                    range.Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;
                }

                int num = 1;
                decimal totalNet = 0;
                foreach (var p in payrolls)
                {
                    row++;
                    ws.Cells[row, 1].Value = num++;
                    ws.Cells[row, 2].Value = p.EmployeeName;
                    ws.Cells[row, 3].Value = p.WorkedHours;
```

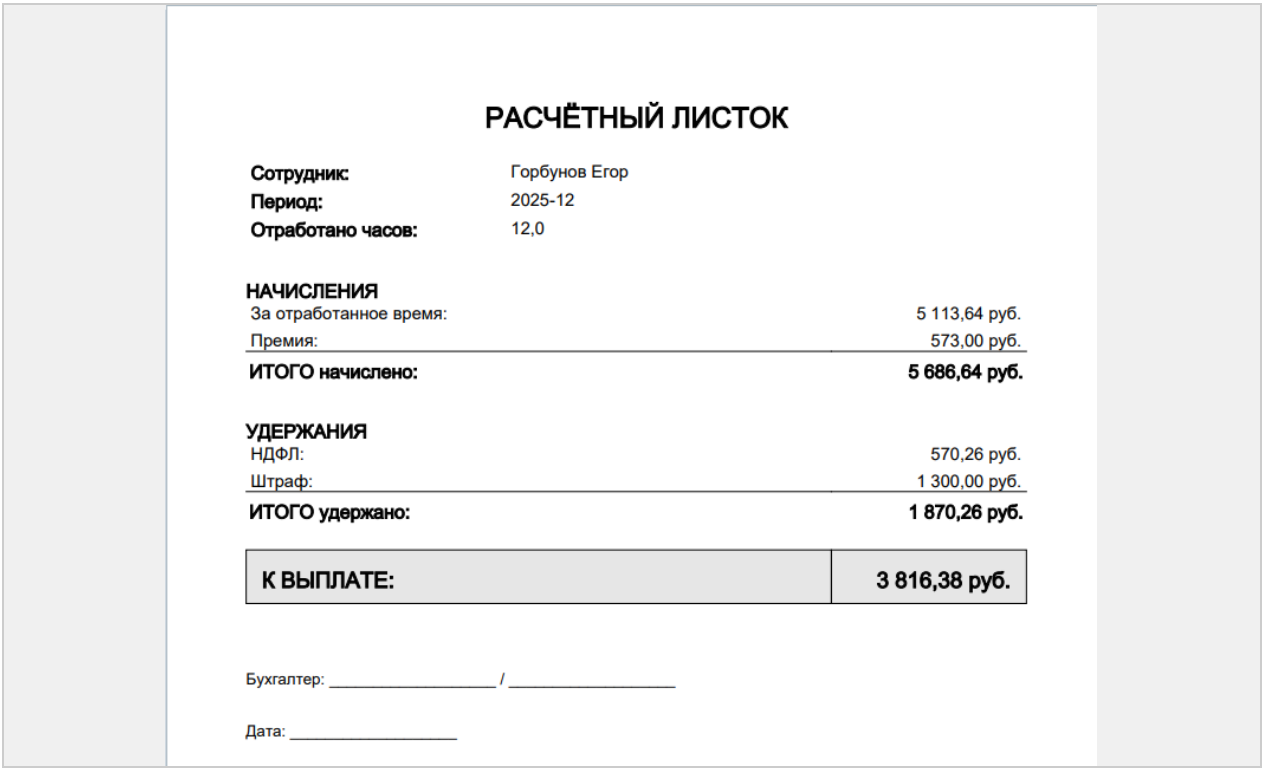
```
                    ws.Cells[row, 4].Value = p.BaseSalary;
                    ws.Cells[row, 5].Value = p.Bonus;
                    ws.Cells[row, 6].Value = p.Penalty;
                    ws.Cells[row, 7].Value = p.TotalDeductions;
                    ws.Cells[row, 8].Value = p.NetSalary;

                    ws.Cells[row, 3].Style.Numberformat.Format = "#,##0.0";
                    ws.Cells[row, 4, row, 8].Style.Numberformat.Format = "#,##0.00";

                    totalNet += p.NetSalary;
                }

                row++;
                ws.Cells[row, 1, row, 7].Merge = true;
                ws.Cells[row, 1].Value = "ИТОГО:";
                ws.Cells[row, 1].Style.Font.Bold = true;
                ws.Cells[row, 1].Style.HorizontalAlignment = ExcelHorizontalAlignment.Right;
                ws.Cells[row, 8].Value = totalNet;
                ws.Cells[row, 8].Style.Font.Bold = true;
                ws.Cells[row, 8].Style.Numberformat.Format = "#,##0.00";

                ws.Cells[4, 1, row, 8].AutoFitColumns();
                ws.Column(2).Width = 30;

                using (var range = ws.Cells[4, 1, row, 8])
                {
                    range.Style.Border.Top.Style = ExcelBorderStyle.Thin;
                    range.Style.Border.Left.Style = ExcelBorderStyle.Thin;
                    range.Style.Border.Right.Style = ExcelBorderStyle.Thin;
                    range.Style.Border.Bottom.Style = ExcelBorderStyle.Thin;
                }

                package.SaveAs(new FileInfo(filePath));
            }
        }
    }
}
```

5.4. Скриншоты работы модуля

[СКРИНШОТ: Форма выбора параметров отчета]



[СКРИНШОТ: Пример сгенерированного PDF расчетного листка]



РАСЧЁТНЫЙ ЛИСТОК

Сотрудник: Горбунов Егор
Период: 2025-12
Отработано часов: 12,0

НАЧИСЛЕНИЯ

За отработанное время:	5 113,64 руб.
Премия:	573,00 руб.
ИТОГО начислено:	**5 686,64 руб.**

УДЕРЖАНИЯ

НДФЛ:	570,26 руб.
Штраф:	1 300,00 руб.
ИТОГО удержано:	**1 870,26 руб.**

К ВЫПЛАТЕ:	**3 816,38 руб.**

Бухгалтер: _____ / _____

Дата: _____

[СКРИНШОТ: Пример Excel-отчета по фонду оплаты труда]

ЗАРПЛАТНАЯ ВЕДОМОСТЬ

Период: 2025-12

№	Сотрудник	Часы	Начислено	Премия	Штраф	Удержания	К выплате
1	Васильева Анна	0,0	0,00	0,00	0,00	0,00	0,00
2	Власова Анастасия	10,0	1 931,82	5 000,00	2 500,00	3 076,14	3 855,68
3	Горбунов Егор	12,0	5 113,64	573,00	1 300,00	1 870,26	3 816,38
4	Зайцев Дмитрий	0,0	0,00	0,00	0,00	0,00	0,00
5	Иванов Иван	40,0	11 363,64	0,00	40,00	1 512,07	9 851,56
6	Кузнецова Мария	40,0	12 500,00	0,00	40,00	1 659,80	10 840,20
7	Морозов Алексей	0,0	0,00	0,00	0,00	0,00	0,00
8	Николаев Николай	40,0	11 363,64	0,00	40,00	1 512,07	9 851,56
9	Петров Пётр	40,0	10 227,27	0,00	40,00	1 364,35	8 862,93
10	Сидоров Сидор	40,0	13 636,36	0,00	40,00	1 807,53	11 828,84
11	Соколова Елена	0,0	0,00	0,00	0,00	0,00	0,00
12	Фёдорова Ольга	0,0	0,00	0,00	0,00	0,00	0,00
						ИТОГО:	**58 907,15**

6. Модуль администрирования (Administration)

6.1. Описание модуля

Назначение: управление пользователями, справочниками и настройками системы.

Функции модуля:

- Создание и удаление пользователей
- Управление справочником отделов

- Управление справочником должностей
- Ведение производственного календаря
- Настройка прав доступа
- Настройка параметров работы системы

6.2. Листинг страницы AdminPage.xaml.cs

```
using System;
using System.Windows;
using System.Windows.Controls;
using PayrollSystem.Models;
using PayrollSystem.Services;

namespace PayrollSystem.Views
{
    public partial class AdminPage : Page
    {
        private readonly DatabaseService _db;

        public AdminPage()
        {
            InitializeComponent();
            _db = new DatabaseService();
            InitializeCalendarComboBoxes();
            cmbRole.SelectedIndex = 1;
            LoadAllData();
        }

        private void InitializeCalendarComboBoxes()
        {
            for (int year = 2020; year <= DateTime.Now.Year + 2; year++)
            {
                cmbCalendarYear.Items.Add(year);
            }
            cmbCalendarYear.SelectedItem = DateTime.Now.Year;

            cmbCalendarMonth.Items.Add(new { Value = 1, Name = "Январь" });
            cmbCalendarMonth.Items.Add(new { Value = 2, Name = "Февраль" });
            cmbCalendarMonth.Items.Add(new { Value = 3, Name = "Март" });
            cmbCalendarMonth.Items.Add(new { Value = 4, Name = "Апрель" });
            cmbCalendarMonth.Items.Add(new { Value = 5, Name = "Май" });
            cmbCalendarMonth.Items.Add(new { Value = 6, Name = "Июнь" });
            cmbCalendarMonth.Items.Add(new { Value = 7, Name = "Июль" });
            cmbCalendarMonth.Items.Add(new { Value = 8, Name = "Август" });
            cmbCalendarMonth.Items.Add(new { Value = 9, Name = "Сентябрь" });
            cmbCalendarMonth.Items.Add(new { Value = 10, Name = "Октябрь" });
            cmbCalendarMonth.Items.Add(new { Value = 11, Name = "Ноябрь" });
            cmbCalendarMonth.Items.Add(new { Value = 12, Name = "Декабрь" });
            cmbCalendarMonth.DisplayMemberPath = "Name";
            cmbCalendarMonth.SelectedValuePath = "Value";
            cmbCalendarMonth.SelectedIndex = DateTime.Now.Month - 1;
        }

        private void LoadAllData()
        {
            LoadUsers();
            LoadDepartments();
            LoadPositions();
            LoadCalendar();
        }

        #region Production Calendar

        private void LoadCalendar()
        {
            try
            {
                int? year = cmbCalendarYear.SelectedItem as int?;
                dgCalendar.ItemsSource = _db.GetProductionCalendar(year);
            }
```

```
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка загрузки календаря: {ex.Message}", "Ошибка",
                    MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }

        private void CmbCalendarYear_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            LoadCalendar();
        }

        private void BtnSaveCalendar_Click(object sender, RoutedEventArgs e)
        {
            if (cmbCalendarYear.SelectedItem == null || cmbCalendarMonth.SelectedValue == null)
            {
                MessageBox.Show("Выберите год и месяц", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
                return;
            }

            if (!decimal.TryParse(txtNormHours.Text, out decimal hours) || hours <= 0)
            {
                MessageBox.Show("Введите корректное количество часов", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
                return;
            }

            try
            {
                int year = (int)cmbCalendarYear.SelectedItem;
                int month = (int)cmbCalendarMonth.SelectedValue;
                _db.SaveProductionCalendar(year, month, hours);
                MessageBox.Show("Сохранено!", "Готово", MessageBoxButton.OK, MessageBoxImage.Information);
                txtNormHours.Text = "";
                LoadCalendar();
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка: {ex.Message}", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        #endregion

        #region Users

        private void LoadUsers()
        {
            try
            {
                dgUsers.ItemsSource = _db.GetAllUsers();
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка загрузки пользователей: {ex.Message}", "Ошибка",
                    MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }

        private void BtnAddUser_Click(object sender, RoutedEventArgs e)
        {
            string username = txtUsername.Text.Trim();
            string password = txtPassword.Text.Trim();
            string role = (cmbRole.SelectedItem as ComboBoxItem)?.Content?.ToString();

            if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password) ||
string.IsNullOrEmpty(role))
            {
                MessageBox.Show("Заполните все поля", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
                return;
            }

            try
            {
```

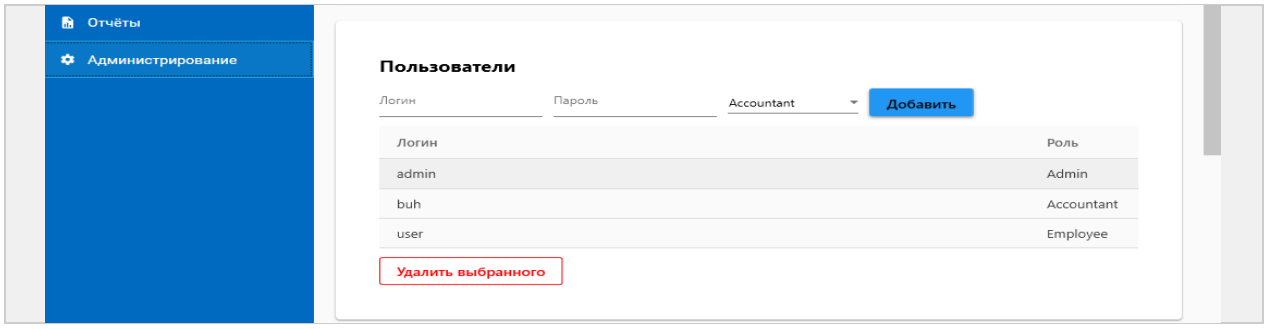
```
                _db.AddUser(new User { Username = username, Password = password, Role = role });
                txtUsername.Text = "";
                txtPassword.Text = "";
                LoadUsers();
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка: {ex.Message}", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void BtnDeleteUser_Click(object sender, RoutedEventArgs e)
        {
            if (dgUsers.SelectedItem is User user)
            {
                if (user.UserID == 1)
                {
                    MessageBox.Show("Нельзя удалить главного администратора", "Внимание",
                        MessageBoxButton.OK, MessageBoxImage.Warning);
                    return;
                }

                if (MessageBox.Show($"Удалить пользователя {user.Username}?", "Подтверждение",
                    MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)
                {
                    try
                    {
                        _db.DeleteUser(user.UserID);
                        LoadUsers();
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($"Ошибка: {ex.Message}", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
                    }
                }
            }
            else
            {
                MessageBox.Show("Выберите пользователя", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
            }
        }

        #endregion

        #region Departments

        private void LoadDepartments()
        {
            try
            {
                dgDepartments.ItemsSource = _db.GetAllDepartments();
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка загрузки отделов: {ex.Message}", "Ошибка",
                    MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }

        private void BtnAddDepartment_Click(object sender, RoutedEventArgs e)
        {
            string name = txtDepartmentName.Text.Trim();
            if (string.IsNullOrEmpty(name))
            {
                MessageBox.Show("Введите название отдела", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
                return;
            }

            try
            {
                _db.AddDepartment(new Department { DepartmentName = name });
                txtDepartmentName.Text = "";
                LoadDepartments();
```

```
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка: {ex.Message}", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void BtnDeleteDepartment_Click(object sender, RoutedEventArgs e)
        {
            if (dgDepartments.SelectedItem is Department dept)
            {
                if (MessageBox.Show($"Удалить отдел \"{dept.DepartmentName}\"?", "Подтверждение",
                    MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)
                {
                    try
                    {
                        _db.DeleteDepartment(dept.DepartmentID);
                        LoadDepartments();
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($"Ошибка: {ex.Message}\n\nВозможно, есть сотрудники в этом
отделе.",
                            "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
                    }
                }
            }
            else
            {
                MessageBox.Show("Выберите отдел", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
            }
        }

        #endregion

        #region Positions

        private void LoadPositions()
        {
            try
            {
                dgPositions.ItemsSource = _db.GetAllPositions();
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка загрузки должностей: {ex.Message}", "Ошибка",
                    MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }

        private void BtnAddPosition_Click(object sender, RoutedEventArgs e)
        {
            string name = txtPositionName.Text.Trim();
            if (string.IsNullOrEmpty(name))
            {
                MessageBox.Show("Введите название должности", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
                return;
            }

            if (!decimal.TryParse(txtPositionSalary.Text, out decimal salary))
            {
                MessageBox.Show("Введите корректный оклад", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
                return;
            }

            try
            {
                _db.AddPosition(new Position { PositionName = name, BaseSalary = salary });
                txtPositionName.Text = "";
                txtPositionSalary.Text = "";
                LoadPositions();
            }
            catch (Exception ex)
```

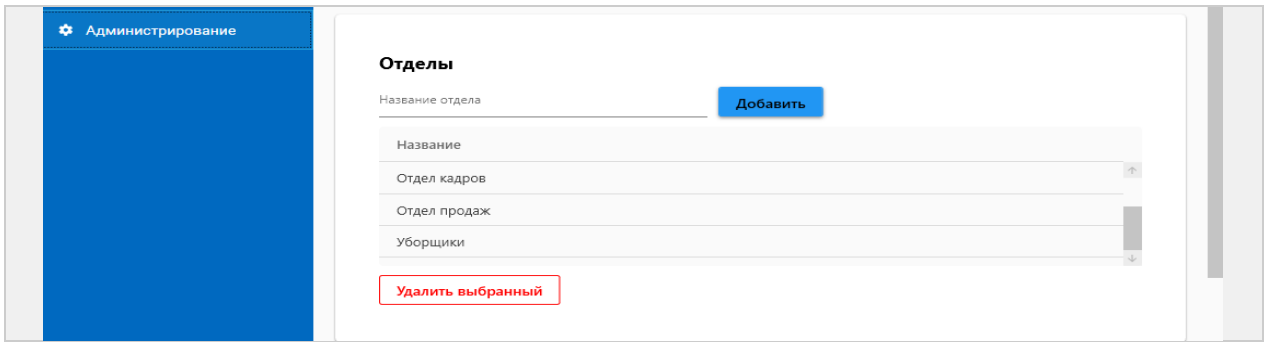
```
            {
                MessageBox.Show($"Ошибка: {ex.Message}", "Ошибка", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void BtnDeletePosition_Click(object sender, RoutedEventArgs e)
        {
            if (dgPositions.SelectedItem is Position pos)
            {
                if (MessageBox.Show($"Удалить должность \"{pos.PositionName}\"?", "Подтверждение",
                    MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)
                {
                    try
                    {
                        _db.DeletePosition(pos.PositionID);
                        LoadPositions();
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($"Ошибка: {ex.Message}\n\nВозможно, есть сотрудники на этой
должности.",
                            "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
                    }
                }
            }
            else
            {
                MessageBox.Show("Выберите должность", "Внимание", MessageBoxButton.OK,
MessageBoxImage.Warning);
            }
        }

        #endregion
    }
}
```

6.3. Скриншоты работы модуля

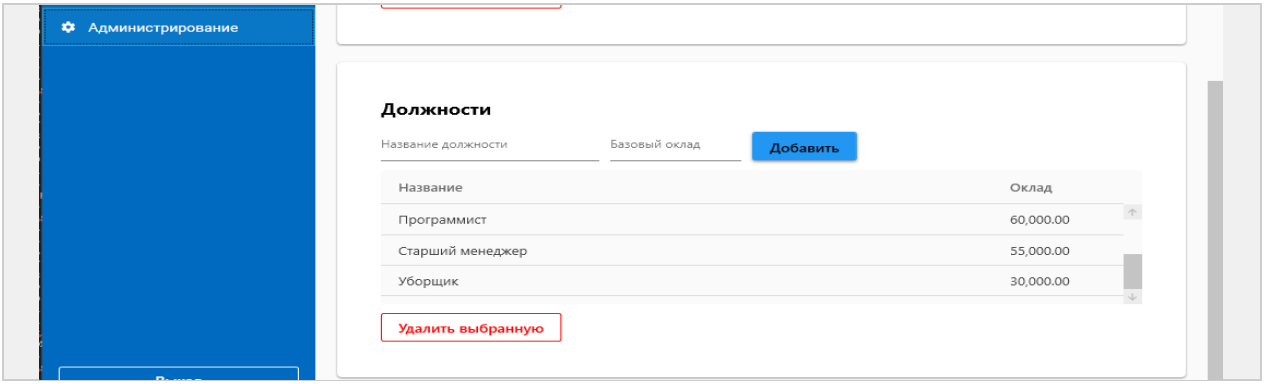
[СКРИНШОТ: Вкладка управления пользователями]



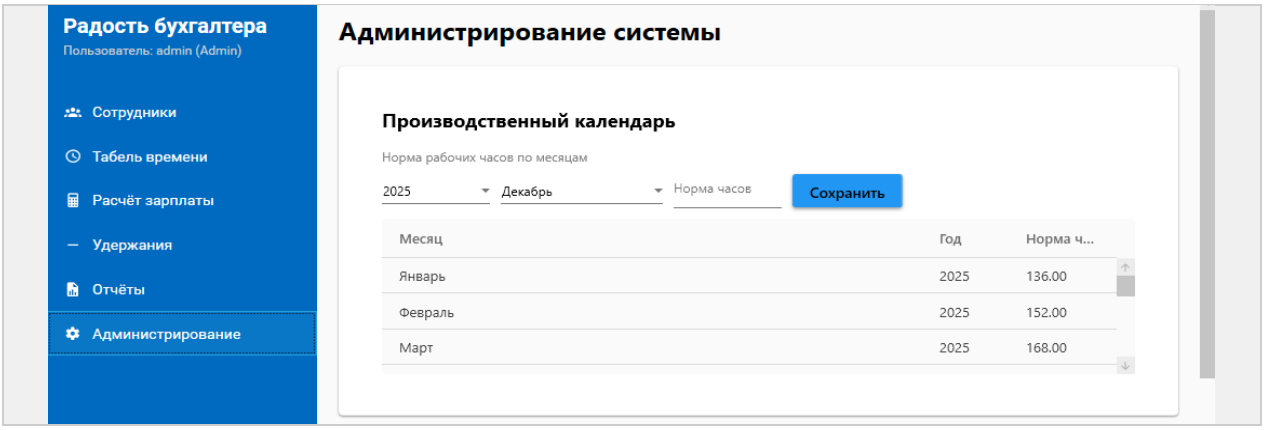
[СКРИНШОТ: Вкладка управления отделами]



[СКРИНШОТ: Вкладка управления должностями]



[СКРИНШОТ: Вкладка управления производственным календарем]



7. Сервис работы с базой данных

7.1. Описание

DatabaseService.cs содержит методы для работы с базой данных SQL Server из пакета EntityFramewirk, обращаясь к его контексту в файле PayrollDbContext.cs. Обеспечивает выполнение операций создания, чтения, обновления и удаления данных (CRUD) для всех модулей системы.

7.2. Листинг DatabaseService.cs

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using PayrollSystem.Data;
using PayrollSystem.Models;

namespace PayrollSystem.Services
{
    public class DatabaseService
    {
        #region Employees

        public List<Employee> GetAllEmployees()
        {
            using (var db = new PayrollDbContext())
            {
                return db.Employees
                    .Include(e => e.Position)
                    .Include(e => e.Department)
                    .OrderBy(e => e.LastName)
                    .ThenBy(e => e.FirstName)
                    .ToList();
```

```
        }
    }

    public void AddEmployee(Employee employee)
    {
        using (var db = new PayrollDbContext())
        {
            db.Employees.Add(employee);
            db.SaveChanges();
        }
    }

    public void UpdateEmployee(Employee employee)
    {
        using (var db = new PayrollDbContext())
        {
            db.Entry(employee).State = EntityState.Modified;
            db.SaveChanges();
        }
    }

    public void DeleteEmployee(int id)
    {
        using (var db = new PayrollDbContext())
        {
            var employee = db.Employees.Find(id);
            if (employee != null)
            {
                db.Employees.Remove(employee);
                db.SaveChanges();
            }
        }
    }

    public decimal GetEmployeeSalary(int employeeId)
    {
        using (var db = new PayrollDbContext())
        {
            return db.Employees.Find(employeeId)?.BaseSalary ?? 0;
        }
    }

    #endregion

    #region Positions

    public List<Position> GetAllPositions()
    {
        using (var db = new PayrollDbContext())
        {
            return db.Positions.OrderBy(p => p.PositionName).ToList();
        }
    }

    public void AddPosition(Position position)
    {
        using (var db = new PayrollDbContext())
        {
            db.Positions.Add(position);
            db.SaveChanges();
        }
    }

    public void DeletePosition(int id)
    {
        using (var db = new PayrollDbContext())
        {
            var position = db.Positions.Find(id);
            if (position != null)
            {
                db.Positions.Remove(position);
                db.SaveChanges();
            }
        }
    }

    #endregion
```

```
        #region Departments

        public List<Department> GetAllDepartments()
        {
            using (var db = new PayrollDbContext())
            {
                return db.Departments.OrderBy(d => d.DepartmentName).ToList();
            }
        }

        public void AddDepartment(Department department)
        {
            using (var db = new PayrollDbContext())
            {
                db.Departments.Add(department);
                db.SaveChanges();
            }
        }

        public void DeleteDepartment(int id)
        {
            using (var db = new PayrollDbContext())
            {
                var department = db.Departments.Find(id);
                if (department != null)
                {
                    db.Departments.Remove(department);
                    db.SaveChanges();
                }
            }
        }

        #endregion

        #region Timesheet

        public List<Timesheet> GetTimesheets(int? employeeId = null, DateTime? startDate = null, DateTime?
endDate = null)
        {
            using (var db = new PayrollDbContext())
            {
                var query = db.Timesheets.Include(t => t.Employee).AsQueryable();

                if (employeeId.HasValue)
                    query = query.Where(t => t.EmployeeID == employeeId.Value);
                if (startDate.HasValue)
                    query = query.Where(t => t.DateWorked >= startDate.Value);
                if (endDate.HasValue)
                    query = query.Where(t => t.DateWorked <= endDate.Value);

                return query.OrderByDescending(t => t.DateWorked).ToList();
            }
        }

        public void AddTimesheet(Timesheet timesheet)
        {
            using (var db = new PayrollDbContext())
            {
                db.Timesheets.Add(timesheet);
                db.SaveChanges();
            }
        }

        public void DeleteTimesheet(int id)
        {
            using (var db = new PayrollDbContext())
            {
                var timesheet = db.Timesheets.Find(id);
                if (timesheet != null)
                {
                    db.Timesheets.Remove(timesheet);
                    db.SaveChanges();
                }
            }
        }
```

```
        public decimal GetWorkedHoursForPeriod(int employeeId, int year, int month)
        {
            using (var db = new PayrollDbContext())
            {
                return db.Timesheets
                    .Where(t => t.EmployeeID == employeeId &&
                                t.DateWorked.Year == year &&
                                t.DateWorked.Month == month)
                    .Sum(t => (decimal?)t.Hours) ?? 0;
            }
        }

        #endregion

        #region Production Calendar

        public List<ProductionCalendar> GetProductionCalendar(int? year = null)
        {
            using (var db = new PayrollDbContext())
            {
                var query = db.ProductionCalendars.AsQueryable();

                if (year.HasValue)
                    query = query.Where(pc => pc.Year == year.Value);

                return query.OrderByDescending(pc => pc.Year).ThenBy(pc => pc.Month).ToList();
            }
        }

        public decimal GetNormHours(int year, int month)
        {
            using (var db = new PayrollDbContext())
            {
                return db.ProductionCalendars
                    .FirstOrDefault(pc => pc.Year == year && pc.Month == month)?.WorkingHours ?? 168;
            }
        }

        public decimal GetNormHoursByPeriod(string period)
        {
            var parts = period.Split('-');
            return GetNormHours(int.Parse(parts[0]), int.Parse(parts[1]));
        }

        public void SaveProductionCalendar(int year, int month, decimal hours)
        {
            using (var db = new PayrollDbContext())
            {
                var existing = db.ProductionCalendars.FirstOrDefault(pc => pc.Year == year && pc.Month ==
month);

                if (existing != null)
                {
                    existing.WorkingHours = hours;
                }
                else
                {
                    db.ProductionCalendars.Add(new ProductionCalendar { Year = year, Month = month,
WorkingHours = hours });
                }

                db.SaveChanges();
            }
        }

        #endregion

        #region Deductions

        public List<Deduction> GetAllDeductions()
        {
            using (var db = new PayrollDbContext())
            {
                return db.Deductions.ToList();
            }
        }
```

```
        public void AddDeduction(Deduction deduction)
        {
            using (var db = new PayrollDbContext())
            {
                db.Deductions.Add(deduction);
                db.SaveChanges();
            }
        }

        public void DeleteDeduction(int id)
        {
            using (var db = new PayrollDbContext())
            {
                var deduction = db.Deductions.Find(id);
                if (deduction != null)
                {
                    db.Deductions.Remove(deduction);
                    db.SaveChanges();
                }
            }
        }

        #endregion

        #region Payroll

        public List<Payroll> GetPayrolls(string period = null)
        {
            using (var db = new PayrollDbContext())
            {
                var query = db.Payrolls
                    .Include(p => p.Employee)
                    .Include(p => p.PayrollDeductions)
                    .AsQueryable();

                if (!string.IsNullOrEmpty(period))
                    query = query.Where(p => p.Period == period);

                return query.OrderBy(p => p.Employee.LastName).ToList();
            }
        }

        public Payroll GetPayrollForEmployee(int employeeId, string period)
        {
            using (var db = new PayrollDbContext())
            {
                return db.Payrolls
                    .Include(p => p.Employee)
                    .Include(p => p.PayrollDeductions)
                    .FirstOrDefault(p => p.EmployeeID == employeeId && p.Period == period);
            }
        }

        public List<PayrollDeduction> GetPayrollDeductions(int payrollId)
        {
            using (var db = new PayrollDbContext())
            {
                return db.PayrollDeductions
                    .Include(pd => pd.Deduction)
                    .Where(pd => pd.PayrollID == payrollId)
                    .ToList();
            }
        }

        public void CalculatePayrollForAll(string period)
        {
            var parts = period.Split('-');
            int year = int.Parse(parts[0]);
            int month = int.Parse(parts[1]);
            decimal normHours = GetNormHours(year, month);

            foreach (var emp in GetAllEmployees())
            {
                decimal workedHours = GetWorkedHoursForPeriod(emp.EmployeeID, year, month);
                var existing = GetPayrollForEmployee(emp.EmployeeID, period);
                decimal bonus = existing?.Bonus ?? 0;
                decimal penalty = existing?.Penalty ?? 0;
```

```
                CalculateAndSavePayroll(emp, period, normHours, workedHours, bonus, penalty);
            }
        }

        private void CalculateAndSavePayroll(Employee emp, string period, decimal normHours, decimal
workedHours, decimal bonus, decimal penalty)
        {
            decimal hourlyRate = normHours > 0 ? emp.BaseSalary / normHours : 0;
            decimal baseSalary = workedHours <= normHours
                ? hourlyRate * workedHours
                : emp.BaseSalary + (workedHours - normHours) * hourlyRate * 1.5m;

            decimal taxBase = Math.Max(0, baseSalary + bonus - penalty);
            decimal tax = taxBase * 0.13m;
            decimal netSalary = taxBase - tax;

            using (var db = new PayrollDbContext())
            {
                var payroll = db.Payrolls.FirstOrDefault(p => p.EmployeeID == emp.EmployeeID && p.Period
== period);

                if (payroll != null)
                {
                    payroll.BaseSalary = baseSalary;
                    payroll.WorkedHours = workedHours;
                    payroll.Bonus = bonus;
                    payroll.Penalty = penalty;
                    payroll.NetSalary = netSalary;
                }
                else
                {
                    payroll = new Payroll
                    {
                        EmployeeID = emp.EmployeeID,
                        Period = period,
                        BaseSalary = baseSalary,
                        WorkedHours = workedHours,
                        Bonus = bonus,
                        Penalty = penalty,
                        NetSalary = netSalary
                    };
                    db.Payrolls.Add(payroll);
                }

                db.SaveChanges();
                SavePayrollDeductions(db, payroll.PayrollID, tax, penalty);
            }
        }

        public void UpdatePayrollBonusAndPenalty(int payrollId, decimal bonus, decimal penalty)
        {
            using (var db = new PayrollDbContext())
            {
                var payroll = db.Payrolls.Find(payrollId);
                if (payroll == null) return;

                decimal taxBase = Math.Max(0, payroll.BaseSalary + bonus - penalty);
                decimal tax = taxBase * 0.13m;

                payroll.Bonus = bonus;
                payroll.Penalty = penalty;
                payroll.NetSalary = taxBase - tax;

                db.SaveChanges();
                SavePayrollDeductions(db, payrollId, tax, penalty);
            }
        }

        private void SavePayrollDeductions(PayrollDbContext db, int payrollId, decimal tax, decimal
penalty)
        {
            var oldDeductions = db.PayrollDeductions.Where(pd => pd.PayrollID == payrollId);
            db.PayrollDeductions.RemoveRange(oldDeductions);

            db.PayrollDeductions.Add(new PayrollDeduction { PayrollID = payrollId, DeductionID = 1, Amount
= tax });
```

```
            if (penalty > 0)
                db.PayrollDeductions.Add(new PayrollDeduction { PayrollID = payrollId, DeductionID = 2,
Amount = penalty });

            db.SaveChanges();
        }

        #endregion

        #region Reports

        public decimal GetTotalPayrollForPeriod(string period)
        {
            using (var db = new PayrollDbContext())
            {
                return db.Payrolls.Where(p => p.Period == period).Sum(p => (decimal?)p.NetSalary) ?? 0;
            }
        }

        public int GetEmployeeCount()
        {
            using (var db = new PayrollDbContext())
            {
                return db.Employees.Count();
            }
        }

        #endregion

        #region Users

        public User Login(string username, string password)
        {
            using (var db = new PayrollDbContext())
            {
                return db.Users.FirstOrDefault(u => u.Username == username && u.Password == password);
            }
        }

        public List<User> GetAllUsers()
        {
            using (var db = new PayrollDbContext())
            {
                return db.Users.ToList();
            }
        }

        public void AddUser(User user)
        {
            using (var db = new PayrollDbContext())
            {
                db.Users.Add(user);
                db.SaveChanges();
            }
        }

        public void DeleteUser(int id)
        {
            using (var db = new PayrollDbContext())
            {
                var user = db.Users.Find(id);
                if (user != null)
                {
                    db.Users.Remove(user);
                    db.SaveChanges();
                }
            }
        }

        #endregion
    }
}
```

7.3. Листинг PayrollDbContext.cs

```
using System.Data.Entity;
using PayrollSystem.Models;

namespace PayrollSystem.Data
{
    public class PayrollDbContext : DbContext
    {
        public PayrollDbContext() : base("name=PayrollConnection")
        {
            Database.SetInitializer<PayrollDbContext>(null);
        }

        public DbSet<Employee> Employees { get; set; }
        public DbSet<Position> Positions { get; set; }
        public DbSet<Department> Departments { get; set; }
        public DbSet<Timesheet> Timesheets { get; set; }
        public DbSet<Payroll> Payrolls { get; set; }
        public DbSet<Deduction> Deductions { get; set; }
        public DbSet<PayrollDeduction> PayrollDeductions { get; set; }
        public DbSet<ProductionCalendar> ProductionCalendars { get; set; }
        public DbSet<User> Users { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Employee>().ToTable("Employees");
            modelBuilder.Entity<Position>().ToTable("Positions");
            modelBuilder.Entity<Department>().ToTable("Departments");
            modelBuilder.Entity<Timesheet>().ToTable("Timesheet");
            modelBuilder.Entity<Payroll>().ToTable("Payroll");
            modelBuilder.Entity<Deduction>().ToTable("Deductions");
            modelBuilder.Entity<PayrollDeduction>().ToTable("PayrollDeductions");
            modelBuilder.Entity<ProductionCalendar>().ToTable("ProductionCalendar");
            modelBuilder.Entity<User>().ToTable("Users");

            modelBuilder.Entity<Employee>().HasKey(e => e.EmployeeID);
            modelBuilder.Entity<Position>().HasKey(p => p.PositionID);
            modelBuilder.Entity<Department>().HasKey(d => d.DepartmentID);
            modelBuilder.Entity<Timesheet>().HasKey(t => t.TimesheetID);
            modelBuilder.Entity<Payroll>().HasKey(p => p.PayrollID);
            modelBuilder.Entity<Deduction>().HasKey(d => d.DeductionID);
            modelBuilder.Entity<PayrollDeduction>().HasKey(pd => pd.PayrollDeductionID);
            modelBuilder.Entity<ProductionCalendar>().HasKey(pc => pc.CalendarID);
            modelBuilder.Entity<User>().HasKey(u => u.UserID);

            modelBuilder.Entity<Employee>()
                .HasRequired(e => e.Position)
                .WithMany(p => p.Employees)
                .HasForeignKey(e => e.PositionID);

            modelBuilder.Entity<Employee>()
                .HasRequired(e => e.Department)
                .WithMany(d => d.Employees)
                .HasForeignKey(e => e.DepartmentID);

            modelBuilder.Entity<Timesheet>()
                .HasRequired(t => t.Employee)
                .WithMany(e => e.Timesheets)
                .HasForeignKey(t => t.EmployeeID);

            modelBuilder.Entity<Payroll>()
                .HasRequired(p => p.Employee)
                .WithMany(e => e.Payrolls)
                .HasForeignKey(p => p.EmployeeID);

            modelBuilder.Entity<PayrollDeduction>()
                .HasRequired(pd => pd.Payroll)
                .WithMany(p => p.PayrollDeductions)
                .HasForeignKey(pd => pd.PayrollID);

            modelBuilder.Entity<PayrollDeduction>()
                .HasRequired(pd => pd.Deduction)
                .WithMany(d => d.PayrollDeductions)
                .HasForeignKey(pd => pd.DeductionID);

            modelBuilder.Entity<Employee>().Ignore(e => e.FullName);
```

```
            modelBuilder.Entity<Employee>().Ignore(e => e.PositionName);
            modelBuilder.Entity<Employee>().Ignore(e => e.DepartmentName);
            modelBuilder.Entity<Timesheet>().Ignore(t => t.EmployeeName);
            modelBuilder.Entity<Payroll>().Ignore(p => p.EmployeeName);
            modelBuilder.Entity<Payroll>().Ignore(p => p.TotalDeductions);
            modelBuilder.Entity<PayrollDeduction>().Ignore(pd => pd.DeductionName);
            modelBuilder.Entity<ProductionCalendar>().Ignore(pc => pc.MonthName);
            modelBuilder.Entity<ProductionCalendar>().Ignore(pc => pc.DisplayText);

            base.OnModelCreating(modelBuilder);
        }
    }
}
```