**ОПИСАНИЕ ОСОБЕННОСТЕЙ РЕАЛИЗАЦИИ КЛИЕНТСКОГО МОДУЛЯ**

**СИСТЕМЫ АВТОМАТИЗИРОВАННОГО СБОРА ДАННЫХ**

**ВЕРСИЯ 1.0**

**СОДЕРЖАНИЕ**

1. ПРИМЕЧАНИЯ 1

2. СТРУКТУРА ПРОЕКТА КЛИЕНТСКОГО МОДУЛЯ 2

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ И ИСПОЛЬЗОВАНИЯ КЛАССОВ 6

**ПРИМЕЧАНИЯ**

*Здесь могут быть примечания по особенностям форматирования кода, именования переменных и функций, использования комментариев и инструментария документации, возможным направлениям рефакторинга (TODO) и т.д.*

**СТРУКТУРА ПРОЕКТА КЛИЕНТСКОГО МОДУЛЯ**

|  |  |
| --- | --- |
| Flight | Класс, представляющий рейс |
| FlightRepository | Класс, представляющий репозиторий для рейсов |
| Ticket | Класс, представляющий билет |
| Ticket Repository | Класс, представляющий репозиторий для билетов |
| User | Класс, представляющий пользователя |
| Log | Класс логирования |

**Основные классы**

|  |  |
| --- | --- |
| IRepository | Интерфейс, представляющий репозиторий |

**Основные интерфейсы**

**Методы и свойства класса Flight**

|  |  |  |  |
| --- | --- | --- | --- |
| Flight | | | |
| Название | | Описание | |
| Поля | | | |
| id | | | Id рейса для хранения в базе данных |
| dispatchCity | | | Город отправки |
| destinationCity | | | Город назначения |
| dispatchTime | | | Время отправки |
| destinationTime | | | Время прибытия |
| places | | | Количество мест |
| Методы | | | |
| Flight() | Конструктор | | |
| Flight (id, dispatchCity, destinationCity, dispatchTime, destinationTime, places) | Конструктор, принимающий все параметры для создания экземпляра | | |
| Id | Геттер для id рейса | | |
| DispatchCity | Геттер и сеттер города отправки | | |
| DestinationCity | Геттер и сеттер города назначения | | |
| DispatchTime | Геттер и сеттер времени отправки | | |
| DestinationTime | Геттер и сеттер времени прибытия | | |
| Places | Геттер и сеттер количества мест | | |

**Методы и свойства класса Ticket**

|  |  |  |  |
| --- | --- | --- | --- |
| Ticket | | | |
| Название | | Описание | |
| Поля | | | |
| id | Id билета для хранения в базе данных | | |
| userId | Id пользователя, купившего этот билет | | |
| flightId | Id рейса | | |
| dispatchDate | День вылета | | |
| price | Цена билета | | |
| place | Номер места | | |
| Методы | | | |
| Ticket () | | | Конструктор |
| Ticket (id, userId, flightId, dispatchDate, price, place) | | | Конструктор, принимающий все параметры для создания экземпляра |
| Ticket (Flight flight, User user, price, place) | | | Конструктор, принимающий экземпляр рейса и пользователя |
| Id | | | Геттер для id билета |
| UserId | | | Геттер и сеттер для id пользователя |
| FlightId | | | Геттер и сеттер для id рейса |
| DispatchDate | | | Геттер и сеттер дня вылета |
| Price | | | Геттер и сеттер цены билеты |
| Place | | | Геттер и сеттер места в самолёте |

**Методы и свойства класса User**

|  |  |  |  |
| --- | --- | --- | --- |
| User | | | |
| Название | | | Описание |
| Поля | | | |
| id | Id пользователя | | |
| surname | Фамилия пользователя | | |
| name | Имя пользователя | | |
| patronymic | Отчество пользователя | | |
| birthDate | Дата рождения пользователя | | |
| address | Адрес проживания пользователя | | |
| telephoneNumber | Номер телефона пользователя | | |
| Методы | | | |
| User () | | Конструктор | |
| User (surname, name, patronymic, birthdate, address, telephoneNumber) | | Конструктор, принимающий все параметры для создания экземпляра | |
| Id | | Геттер для id пользователя | |
| Surname | | Геттер и сеттер для фамилии | |
| Name | | Геттер и сеттер для имени пользователя | |
| Patronymic | | Геттер и сеттер для отчества пользователя | |
| Birthdate | | Геттер и сеттер для даты рождения пользователя | |
| Address | | Геттер и сеттер для адреса проживания пользователя | |
| TelephoneNumber | | Геттер и сеттер для номера телефона пользователя | |

**Методы и свойства интерфейса IRepository**

|  |  |  |
| --- | --- | --- |
| IRepository | | |
| Название | | Описание |
| Методы | | |
| GetList() | Получение всех объектов | |
| GetInfo(id) | Получение одного объекта | |
| Create(T item) | Создание объекта | |
| Update(T item) | Обновление объекта | |
| Delete(id) | Удаление объекта по id | |
| Save() | Сохранение изменений | |

**Методы и свойства класса FlightRepository**

|  |  |  |
| --- | --- | --- |
| FlightRepository | | |
| Название | | Описание |
| Методы | | |
| GetList() | Получение всех объектов | |
| GetInfo(id) | Получение одного объекта | |
| Create(item) | Создание объекта | |
| Update(item) | Обновление объекта | |
| Delete(id) | Удаление объекта по id | |
| Save() | Сохранение изменений | |

**Методы и свойства класса TicketRepository**

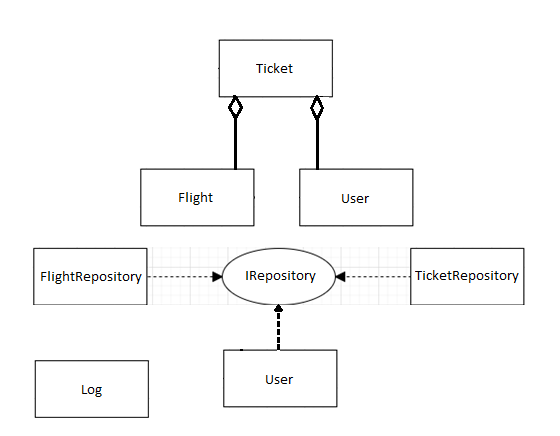
|  |  |  |
| --- | --- | --- |
| TicketRepository | | |
| Название | | Описание |
| Методы | | |
| GetList() | Получение всех объектов | |
| GetInfo(id) | Получение одного объекта | |
| Create(item) | Создание объекта | |
| Update(item) | Обновление объекта | |
| Delete(id) | Удаление объекта по id | |
| Save() | Сохранение изменений | |

**Методы и свойства класса UserRepository**

|  |  |  |
| --- | --- | --- |
| UserRepository | | |
| Название | | Описание |
| Методы | | |
| GetList() | Получение всех объектов | |
| GetInfo(id) | Получение одного объекта | |
| Create(item) | Создание объекта | |
| Update(item) | Обновление объекта | |
| Delete(id) | Удаление объекта по id | |
| Save() | Сохранение изменений | |

**Методы и свойства класса Log**

|  |  |  |  |
| --- | --- | --- | --- |
| Log | | | |
| Название | | Описание | |
| Поля | | | |
| Name | Название | | |
| operation | Суть операции | | |
| Time | Время операции | | |
| Методы | | | |
| Log() | | | конструктор |
| GetLog() | | | Метод для получения отчета |

Общая схема классов уровня логики приложения приведена на Рис.1. ****

**Рисунок 1 – UML-диаграмма классов уровня логики приложения**

**ОСОБЕННОСТИ РЕАЛИЗАЦИИ И ИСПОЛЬЗОВАНИЯ КЛАССОВ И ИНТЕРФЕЙСОВ**

* **Log**

Класс, позволяющий логировать все события и действия пользователя.

Ниже приведен программный код системы

public class Log

{

public Log() { }

private DateTime time;

private string name;

private string operation;

public void GetLog() { }

}

* **Flight**

Класс, описывающий рейс как объект

Ниже приведен программный код системы

class Flight

{

Private static int id;

private City dispatchCity;

private City destinationCity;

private DateTime dispatchTime;

private DateTime destinationTime;

private int places;

public Flight(City dispatchCity, City destinationCity, DateTime dispatchTime, DateTime destinationTime, int places)

{

this.dispatchCity = dispatchCity;

this.destinationCity = destinationCity;

this.dispatchTime = dispatchTime;

this.destinationTime = destinationTime;

this.places = places;

}

public int Id { get => id; }

public City DispatchCity { get => dispatchCity;   
 set => dispatchCity = value; }

public City DestinationCity { get => destinationCity;   
 set => destinationCity = value; }

public DateTime DispatchTime { get => dispatchTime;  
 set => dispatchTime = value; }

public DateTime DestinationTime { get => destinationTime;  
 set => destinationTime = value; }

public int Places { get => places; set => places = value; }

}

* **User**

Класс, описывающий пользователя как объект

Ниже приведен программный код системы

class User

{

private int id;

private string surname;

private string name;

private string patronymic;

private DateTime birthDate;

private string address;

private string telephoneNumber;

public User (string surname, string name, string patronymic, DateTime birthdate, string address, string telephoneNumber)

{

this.surname = surname;

this.name = name;

this.patronymic = patronymic;

this.birthdate = birthdate;

this.address = address;

this.telephoneNumber = telephoneNumber;

}

public int Id { get => id; private set => id = value; }

public string Surname { get => surname; set => surname = value; }

public string Name { get => name; set => name = value; }

public string Patronymic { get => patronymic; set => patronymic = value; }

public DateTime Birthdate { get => birthdate; set => birthdate = value; }

public string Address { get => address; set => address = value; }

public string TelephoneNumber { get => telephoneNumber; set => telephoneNumber = value; }

}

* **Ticket**

Класс, описывающий сеанс

Ниже приведен программный код системы

class Ticket

{

private int id;

private int userId;

private int flightId;

private DateTime dispatchDate;

private int price;

private int place;

public Ticket(int userId, int flightId, DateTime dispatchDate, int price, int place)

{

this.userId = userId;

this.flightId = flightId;

this.dispatchDate = dispatchDate;

this.price = price;

this.place = place;

}

public Ticket(Flight flight, User user, int price, int place)

{

this.userId = user.Id;

this.flightId = flight.Id;

this.dispatchDate = flight.DispatchTime;

this.price = price;

this.place = place;

}

public int Id { get => id; private set => id = value; }

public int UserId { get => userId; set => userId = value; }

public int FlightId { get => flightId; set => flightId = value; }

public DateTime DispatchDate { get => dispatchDate;  
set => dispatchDate = value; }

public int Price { get => price; set => price = value; }

public int Place { get => place; set => place = value; }

}

* **IRepository**

Интерфейс, описывающий репозиторий для хранения данных

Ниже приведен программный код системы

interface IRepository<T> : IDisposable

where T : class

{

IEnumerable<T> GetList();

T GetInfo(int id);

void Create(T item);

void Update(T item);

void Delete(int id);

void Save();

}

* **FlightRepository**

Класс, реализующий интерфейс IRepository для хранения данных о рейсах

Ниже приведен программный код системы

class FlightRepository : IRepository<Flight>

{

private bool disposedValue;

public void Create(Flight item){ }

public void Delete(int id) { }

public Flight GetInfo(int id) { }

public IEnumerable< Flight > GetList() { }

public void Save() { }

public void Update(Flight item) { }

protected virtual void Dispose(bool disposing)

{

if (!disposedValue)

{

if (disposing) { }

disposedValue = true;

}

}

// TODO: override finalizer only if 'Dispose(bool disposing)' has code to free unmanaged resources

// ~FlightRepository()

public void Dispose()

{

Dispose(disposing: true);

GC.SuppressFinalize(this);

}

}

* **TicketRepository**

Класс, реализующий интерфейс IRepository для хранения данных о билетах

Ниже приведен программный код системы

class TicketRepository : IRepository<Ticket>

{

private bool disposedValue;

public void Create(Ticket item){ }

public void Delete(int id) { }

public Ticket GetInfo(int id) { }

public IEnumerable< Ticket > GetList() { }

public void Save() { }

public void Update(Ticket item) { }

protected virtual void Dispose(bool disposing)

{

if (!disposedValue)

{

if (disposing) { }

disposedValue = true;

}

}

// TODO: override finalizer only if 'Dispose(bool disposing)' has code to free unmanaged resources

// ~TicketRepository()

public void Dispose()

{

Dispose(disposing: true);

GC.SuppressFinalize(this);

}

}

* **UserRepository**

Класс, реализующий интерфейс IRepository для хранения данных о пользователях

Ниже приведен программный код системы

class UserRepository : IRepository<User>

{

private bool disposedValue;

public void Create(User item){ }

public void Delete(int id) { }

public User GetInfo(int id) { }

public IEnumerable< User > GetList() { }

public void Save() { }

public void Update(User item) { }

protected virtual void Dispose(bool disposing)

{

if (!disposedValue)

{

if (disposing) { }

disposedValue = true;

}

}

// TODO: override finalizer only if 'Dispose(bool disposing)' has code to free unmanaged resources

// ~UserRepository()

public void Dispose()

{

Dispose(disposing: true);

GC.SuppressFinalize(this);

}

}