ОБЩАЯ АРХИТЕКТУРА ПРИЛОЖЕНИЯ НА ОСНОВЕ СПЕЦИФИКАЦИЙ ТРЕБОВАНИЙ «ИНФОРМАЦИОННАЯ СИСТЕМА ЖЕЛЕЗНОДОРОЖНОГО ВОКЗАЛА»

**1. Архитектурный подход**

* **Платформа: Windows Presentation Foundation (WPF)**:
  + Мощный фреймворк для создания настольных приложений
  + Поддержка современного дизайна и анимации
  + Гибкая система привязки данных
* **Паттерн проектирования: MVVM (Model-View-ViewModel)**:
  + Улучшенное разделение логики и представления
  + Упрощенное тестирование компонентов
  + Легкость поддержки и расширения функционала

**2. Компоненты системы**

**2.1 Уровень представления (Presentation Layer)**

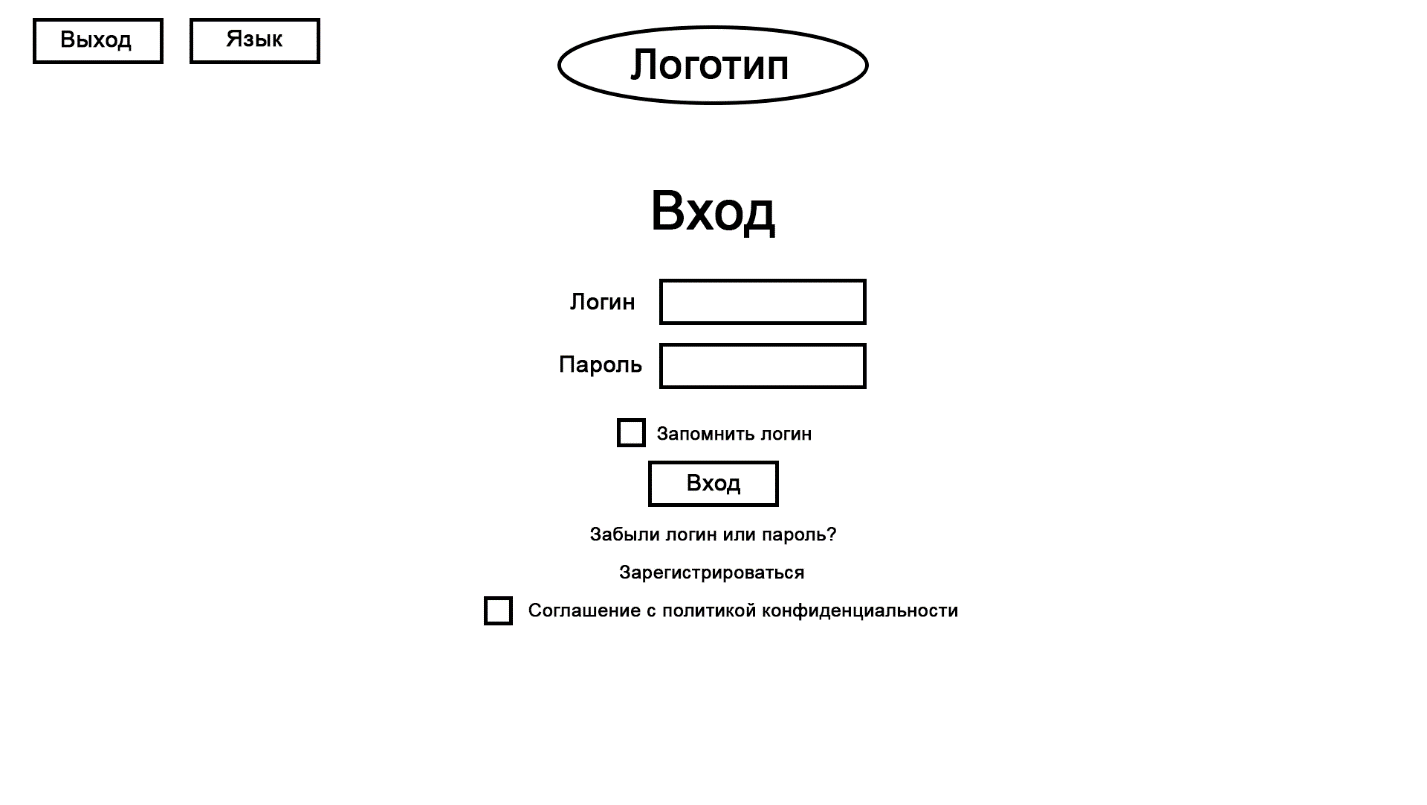
* **Окно входа (Вход в систему)**:
  + Поля ввода логина и пароля
  + Обработка ошибок авторизации
  + Регистрация нового сотрудника
* **Главное окно расписания (Система вокзала)**:
  + Таблица расписания с сортировкой и фильтрацией
  + Кнопки добавления, редактирования, удаления рейсов
  + Модальные окна для ввода/правки информации
  + Контекстное меню
* **Технологии реализации**:
  + XAML для верстки интерфейса
  + Data Binding для связи данных
  + Стили и шаблоны WPF

**2.2 Уровень бизнес-логики (Business Layer)**

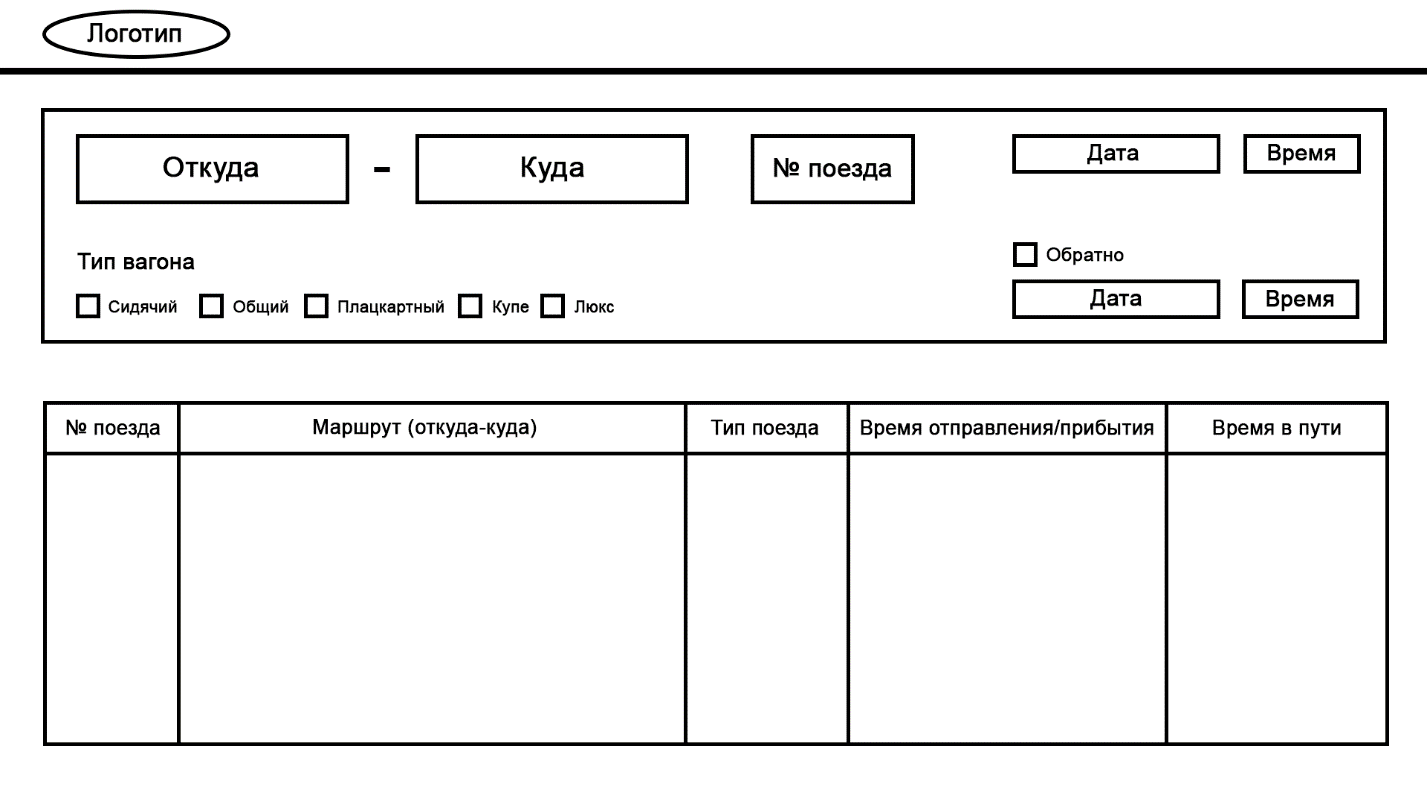
* **Модели данных**:
  + Train - полное описание поезда
  + Schedule - управление расписанием
  + User - модель пользователя системы
* **Сервисы и функционал**:
  + Валидация данных о поездах
  + Бизнес-правила добавления/удаления рейсов
  + Проверка пересечений расписания
  + Логирование действий пользователей
* **Обработка исключительных ситуаций**:
  + Кастомные исключения
  + Graceful error handling
  + Информативные сообщения пользователю

1. МАКЕТ ИНТЕРФЕЙСА ПРОГРАММНОГО ПРОДУКТА

Главная страница приложения железнодорожного вокзала



Страница с расписанием



ОПИСАНИЕ ОСОБЕННОСТЕЙ РЕАЛИЗАЦИИ КЛИЕНТСКОГО МОДУЛЯ СИСТЕМЫ АВТОМАТИЗИРОВАННОГО СБОРА ДАННЫХ

Версия 1.0\_final

# **ПРИМЕЧАНИЯ**

Документация разработана на основе технического задания для информационной системы железнодорожного вокзала

**Назначение модуля**

Клиентский модуль предназначен для автоматизации и оптимизации процессов работы железнодорожного вокзала, включая управление расписанием поездов, обработку данных сотрудников и предоставление информации пассажирам.

**Технические ограничения**

* Программная платформа: Windows.
* Минимальные требования к оборудованию:
  + **Процессор**: Intel/AMD.
  + **ОЗУ**: 1 ГБ.
  + **Хранилище**: 1 ГБ.

**Язык реализации**: C#.

# **СТРУКТУРА ПРОЕКТА КЛИЕНТСКОГО МОДУЛЯ**

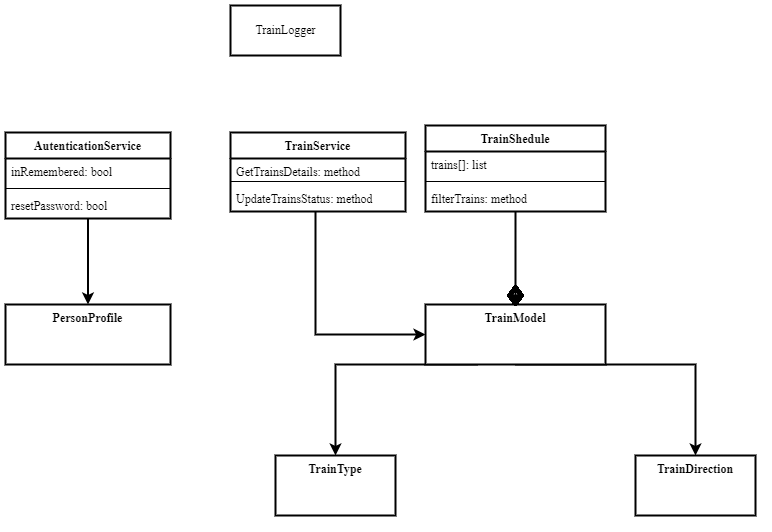
Логика приложения (классы и соответствующие файлы .cs)

|  |  |
| --- | --- |
| TrainModel | Основные характеристики поезда (номер, маршрут, тип вагонов, услуги и т.д.) |
| TrainType | Различные типы поездов (скоростной, пригородный, экспресс и т.д.) |
| TrainDirection | Направления движения поездов (прибытие, отправление) |
| TrainLogger | Логгер (класс, работающий с логом событий и действий пользователя) |
| TrainShedule | Информация о конкретном поезде |
| TrainService | Управление информацией о поездах |
| AuthenticationService | Аутентификация пользователей |
| PersonProfile | Пользователь информационной системы |

Окна и GUI-элементы (классы и соответствующие файлы .cs)

|  |  |
| --- | --- |
| LoginForm.cs | Окно входа |
| RegistrationForm.cs | Форма регистрации сотрудника |
| MainForm.cs | Окно расписания поездов |
| MainLogger.cs | Окно с логом событий |
| TrainShedule.cs | Окно добавления, изменения, удаления поезда из расписания |

Общая схема классов уровня логики приложения приведена на рис. 1



**Рис. 1. – UML-диаграмма классов уровня логики приложения**

# **ОСОБЕННОСТИ РЕАЛИЗАЦИИ И ИСПОЛЬЗОВАНИЯ КЛАССОВ**

// Класс TrainModel

public class TrainModel

{

// Уникальный номер поезда

public int TrainNumber { get; set; }

// Маршрут следования поезда

public string TrainRoute { get; set; }

// Время отправления поезда

public DateTime DepartureTime { get; set; }

// Время прибытия поезда

public DateTime ArrivalTime { get; set; }

// Текущий статус поезда (например, "По расписанию", "Задержка", "Отменен")

public TrainStatus Status { get; set; }

// Список типов вагонов, входящих в состав поезда

public List<TrainCarType> CarTypes { get; set; }

}

// Класс TrainType

public class TrainType

{

// Название типа поезда ("Скоростной", "Пригородный", "Экспресс")

public string Name { get; set; }

// Описание типа поезда

public string Description { get; set; }

}

// Класс TrainDirection

public class TrainDirection

{

// Название направления ("Прибытие", "Отправление")

public string Name { get; set; }

// Флаг, указывающий, является ли данное направление прибытием

public bool IsArrival { get; set; }

}

// Класс TrainSchedule

public class TrainSchedule

{

private List<TrainModel> \_trains;

// Список поездов, входящих в расписание

public List<TrainModel> Trains

{

get { return \_trains; }

}

// Метод для фильтрации списка поездов по дате и направлению

public List<TrainModel> FilterTrains(DateTime date, TrainDirection direction)

{

return \_trains.Where(t => t.DepartureTime.Date == date.Date && t.Direction == direction).ToList();

}

// Метод для сортировки списка поездов по различным критериям

public List<TrainModel> SortTrains(SortCriteria criteria)

{

switch (criteria)

{

case SortCriteria.DepartureTime:

return \_trains.OrderBy(t => t.DepartureTime).ToList();

case SortCriteria.ArrivalTime:

return \_trains.OrderBy(t => t.ArrivalTime).ToList();

case SortCriteria.TrainNumber:

return \_trains.OrderBy(t => t.TrainNumber).ToList();

default:

return \_trains;

}

}

}

// Класс AuthenticationService

public class AuthenticationService

{

// Метод для входа пользователя в систему

public bool Login(string username, string password)

{

// Реализация логики входа

// Возвращает true в случае успешной аутентификации, false - в противном случае

}

// Метод для регистрации нового пользователя

public bool Register(UserCredentials credentials)

{

// Реализация логики регистрации

// Возвращает true в случае успешной регистрации, false - в противном случае

}

// Метод для сброса пароля пользователя

public bool ResetPassword(string username)

{

// Реализация логики сброса пароля

// Возвращает true в случае успешного сброса, false - в противном случае

}

// Метод для проверки, был ли пользователь ранее запомнен системой

public bool IsRemembered()

{

// Реализация логики проверки запомненного состояния

// Возвращает true, если пользователь был запомнен, false - в противном случае

}

}

// Класс PersonProfile представляет профиль пользователя после авторизации

public class PersonProfile

{

// Уникальный идентификатор пользователя

public int UserID { get; set; }

// Имя пользователя

public string FirstName { get; set; }

// Фамилия пользователя

public string LastName { get; set; }

// Электронная почта пользователя

public string Email { get; set; }

// Метод для обновления личных данных пользователя

public bool UpdateProfile(string firstName, string lastName, string email, string phoneNumber)

{

// Реализация логики обновления профиля

// Возвращает true в случае успешного обновления, false - в противном случае

FirstName = firstName;

LastName = lastName;

Email = email;

return true;

}

// Метод для получения полного имени пользователя

public string GetFullName()

{

return $"{FirstName} {LastName}";

}

}

// Класс TrainService

public class TrainService

{

// Метод для получения подробной информации о поезде по его номеру

public TrainModel GetTrainDetails(int trainNumber)

{

// Реализация логики получения информации о поезде

// Возвращает объект TrainModel

}

// Метод для обновления статуса поезда

public bool UpdateTrainStatus(int trainNumber, TrainStatus newStatus)

{

// Реализация логики обновления статуса поезда

// Возвращает true в случае успешного обновления, false - в противном случае

}

// Класс TrainLogger для логирования информации о поездах и действиях пользователей

public class TrainLogger

{

// Список всех логов

private List<LogEntry> \_logEntries;

// Метод для добавления записи в лог

public void LogEvent(LogType type, string message, PersonProfile user = null)

{

LogEntry entry = new LogEntry

{

Timestamp = DateTime.Now,

Type = type,

Message = message,

User = user

};

\_logEntries.Add(entry);

SaveLogToFile(entry);

}

// Метод для получения логов за определенный период

public enum UserRole

{

Администратор,

Диспетчер,

Кассир

}

public enum LogType

{

Information,

Warning,

Error,

EmployeeAction

}

public class LogEntry

{

private const string LogFilePath = "train\_logs.log";

// Константы для типов операций

public const byte LOGIN\_OPERATION = 0x11;

public const byte STATUS\_CHANGE\_OPERATION = 0x12;

public const byte SCHEDULE\_MODIFICATION\_OPERATION = 0x13;

// Статический метод логирования

public static void Log(PersonProfile user, LogType type, string message, byte operationType = 0)

{

try

{

DateTime timestamp = DateTime.Now;

// Логирование в текстовый файл

LogTextEntry(user, type, message, timestamp);

// Если указан тип операции - логируем служебную информацию

if (operationType != 0)

{

LogEmployeeAction(user, timestamp, operationType);

}

}

catch (Exception ex)

{

// Fallback logging to console

Console.WriteLine($"Error log: {ex.Message}");

}

}

// Метод текстового логирования

private static void LogTextEntry(PersonProfile user, LogType type, string message, DateTime timestamp)

{

string logMessage = $"[{timestamp:yyyy-MM-dd HH:mm:ss}] [{type}] {message}";

if (user != null)

{

logMessage += $" (User: {user.Username}, Role: {user.Role})";

}

File.AppendAllText(LogFilePath, logMessage + Environment.NewLine);

}

// Метод служебного логирования действий сотрудника

private static void LogEmployeeAction(PersonProfile user, DateTime timestamp, byte operationType)

{

if (user == null)

{

throw new InvalidOperationException("The user cannot be null to log employee actions");

}

using (FileStream fs = new FileStream(LogFilePath, FileMode.Append, FileAccess.Write))

{

// 8 байт - дата/время в формате FILETIME

byte[] timestampBytes = BitConverter.GetBytes(timestamp.ToFileTime());

fs.Write(timestampBytes, 0, 8);

// 1 байт - тип операции

fs.WriteByte(operationType);

// Преобразование ID сотрудника в байты

string employeeId = user.ID.ToString("D8");

byte idLength = (byte)employeeId.Length;

// 1 байт - длина ID

fs.WriteByte(idLength);

// M байт - ID сотрудника

byte[] idBytes = Encoding.UTF8.GetBytes(employeeId);

fs.Write(idBytes, 0, idBytes.Length);

// 1 байт - роль сотрудника

byte roleCode = ConvertRoleToCode(user.Role);

fs.WriteByte(roleCode);

}

}

// Преобразование роли в код

private static byte ConvertRoleToCode(string role)

{

switch (role)

{

case "Администратор": return 0x01;

case "Диспетчер": return 0x02;

case "Кассир": return 0x03;

default: throw new ArgumentException("Неизвестная роль");

}

}

}**ОПИСАНИЕ CONTROL FLOW ОСНОВНЫХ ЧАСТЕЙ ПРОЕКТА**

1. Аутентификация и авторизация (AuthenticationService)

A[Начало входа] --> B{Ввод корпоративных данных}

B --> C[Проверка учетных данных]

C --> |Успешно| D[Идентификация роли сотрудника]

C --> |Неуспешно| E[Блокировка доступа]

D --> F{Проверка полномочий}

F --> |Доступ разрешен| G[Вход в систему]

F --> |Доступ запрещен| H[Ограничение функционала]

G --> I[Логирование входа]

H --> E

I --> J[Открытие рабочего интерфейса]

2. Управление статусами поездов (TrainService)

A[Запрос изменения статуса] --> B{Проверка полномочий}

B --> |Доступ есть| C[Выбор нового статуса]

B --> |Доступ закрыт| D[Отклонение операции]

C --> E[Подтверждение изменений]

E --> F[Обновление модели поезда]

F --> G[Автоматическое логирование]

G --> H[Уведомление причастных служб]

D --> I[Логирование попытки]

3. Система логирования действий (TrainLogger)

A[Системное событие] --> B{Определение типа события}

B --> |Вход| C[Регистрация аутентификации]

B --> |Изменение статуса| D[Протоколирование деталей]

B --> |Системная ошибка| E[Критический учет]

C --> F[Сохранение метаданных]

D --> F

E --> F

F --> G[Архивация записи]

G --> H[Опциональное уведомление]

# **ФОРМАТЫ БИНАРНЫХ ФАЙЛОВ**

**Файл train\_terminal.dat**

* 4 байта: ID терминала (unsigned int)
* 4 байта: количество символов в названии станции M (unsigned int)
* M\*2 байта: M символов названия станции (Unicode)
* 4 байта: код региона (int)
* 4 байта: количество обслуживаемых платформ N (unsigned int)
* N блоков со структурой:
  + 4 байта: номер платформы P (unsigned int)
  + 2 байта: статус платформы (short)
  + 4 байта: количество символов в описании платформы Q (unsigned int)
  + Q\*2 байта: Q символов описания платформы (Unicode)

**Файл train\_logs.log**

Файл содержит блоки трех типов:

**Блок типа 1 (Событие с поездом):**

* 8 байт: Дата/время события (FILETIME)
* 1 байт: тип события (unsigned char)
  + 0x01 - отправление
  + 0x02 - прибытие
  + 0x03 - задержка
* 1 байт: M - количество символов в номере поезда (unsigned char)
* M байт: номер поезда
* 4 байта: задержка в минутах (int)

**Блок типа 2 (Действия сотрудника):**

* 8 байт: Дата/время события (FILETIME)
* 1 байт: тип операции (unsigned char)
  + 0x11 - вход в систему
  + 0x12 - изменение статуса
  + 0x13 - модификация расписания
* 1 байт: M - количество символов в ID сотрудника (unsigned char)
* M байт: ID сотрудника
* 1 байт: роль сотрудника (unsigned char)

**Блок типа 3 (Системные происшествия):**

* 8 байт: Дата/время события (FILETIME)
* 1 байт: тип происшествия (unsigned char)
  + 0x21 - техническая неисправность
  + 0x22 - сбой в расписании
  + 0x23 - чрезвычайная ситуация
* 1 байт: M - количество символов в описании (unsigned char)
* M байт: описание происшествия
* 2 байта: уровень серьезности (short)

**Файл train\_schedule.dat**

* 4 байта: количество записей о поездах N (unsigned int)
* N блоков со структурой:
  + 4 байта: номер поезда (unsigned int)
  + 8 байт: время отправления (FILETIME)
  + 8 байт: время прибытия (FILETIME)
  + 1 байт: M - количество символов в маршруте (unsigned char)
  + M байт: маршрут следования
  + 2 байта: платформа отправления (short)
  + 2 байта: платформа прибытия (short)
  + 1 байт: статус поезда (unsigned char)

**Файл employee\_records.dat**

* 4 байта: количество записей о сотрудниках N (unsigned int)
* N блоков со структурой:
  + 1 байт: M - количество символов в ID сотрудника (unsigned char)
  + M байт: ID сотрудника
  + 1 байт: N - количество символов в имени (unsigned char)
  + N байт: имя сотрудника
  + 1 байт: K - количество символов в фамилии (unsigned char)
  + K байт: фамилия сотрудника
  + 1 байт: роль сотрудника (unsigned char)
  + 8 байт: дата последнего входа (FILETIME)

