ЛАБОРАТОРНАЯ РАБОТА №2



Курс «Технологии разработки программного обеспечения»

Тема: Проектирование и макетирование программного продукта.

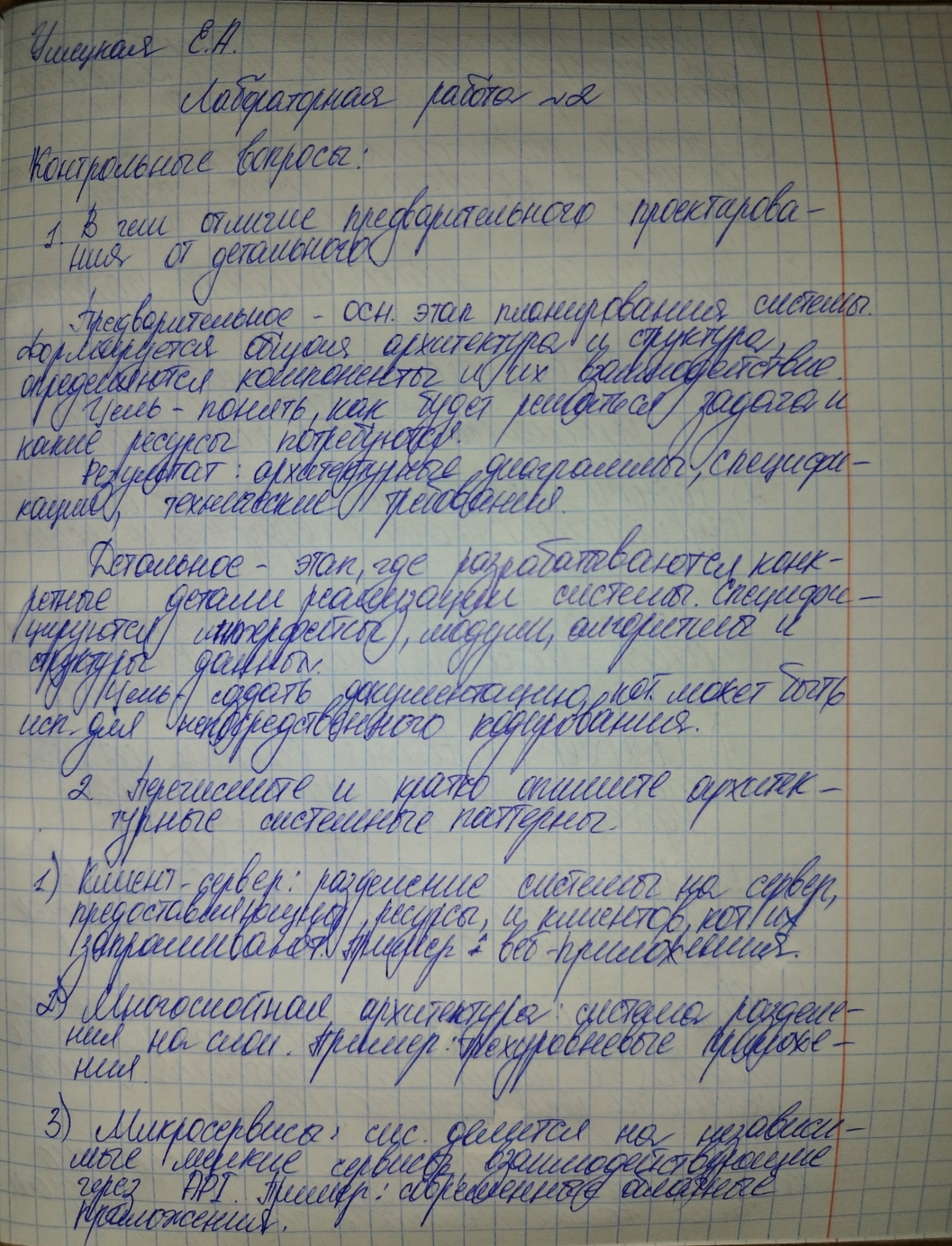
Цель: Научиться проектировать простейшие системы и составлять документацию по проектированию программного продукта.

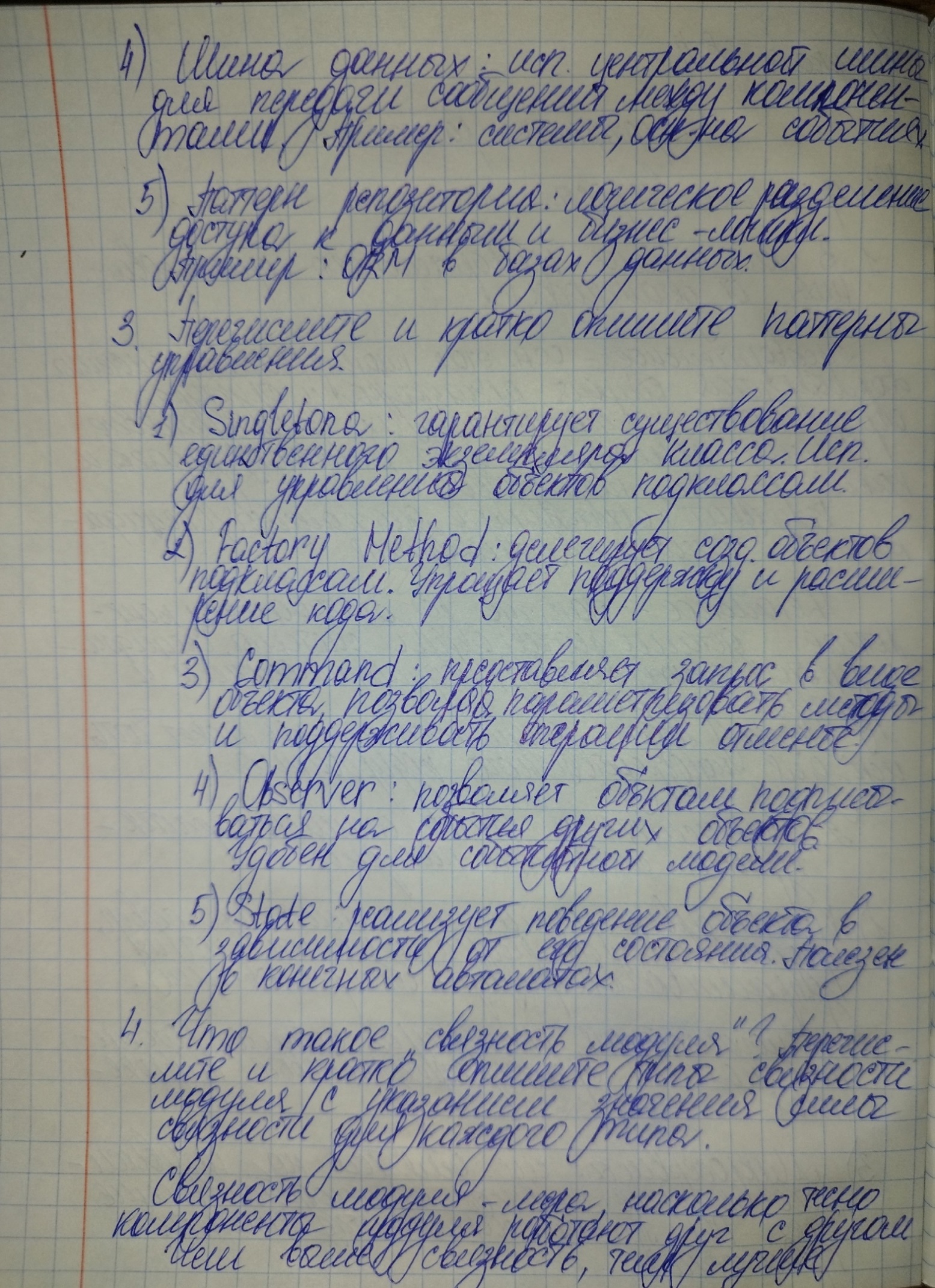
Задание:

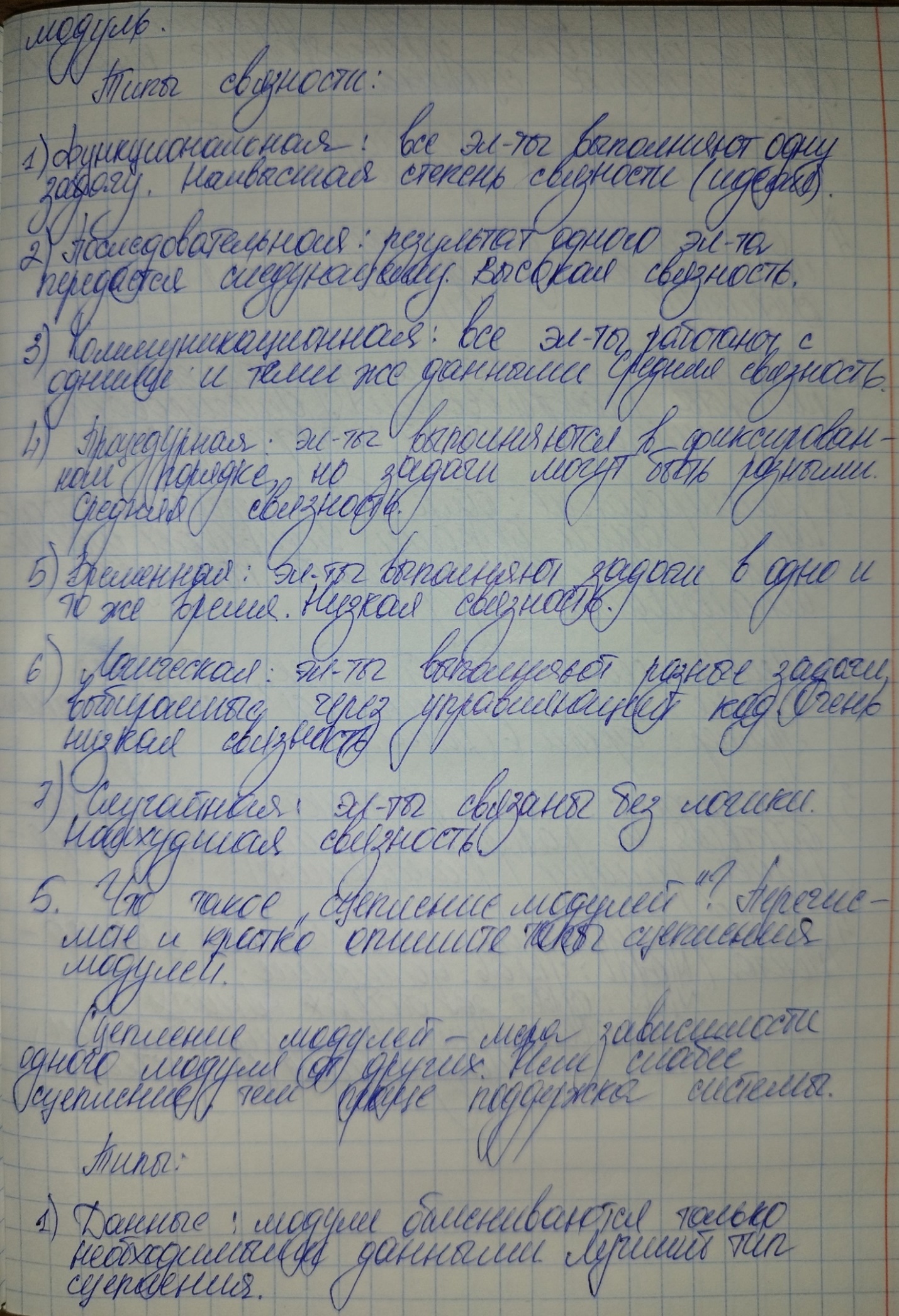
1. Спроектировать общую архитектуру приложения на основе спецификаций требований, составленных в лабораторной работе №1.
2. Разработать макет интерфейса программного продукта для демонстрации заказчику. Проанализировать и учесть его замечания и пожелания.
3. Оформить разделы документа SDD, относящиеся к предварительному проектированию (приложение 2.1).
4. Программно реализовать первую версию спроектированной системы (создать программные модули и закодировать основные классы / структуры и связи между ними).
5. Оформить отчет, включающий ответы на контрольные вопросы, программный код системы и первую версию SDD-документа.

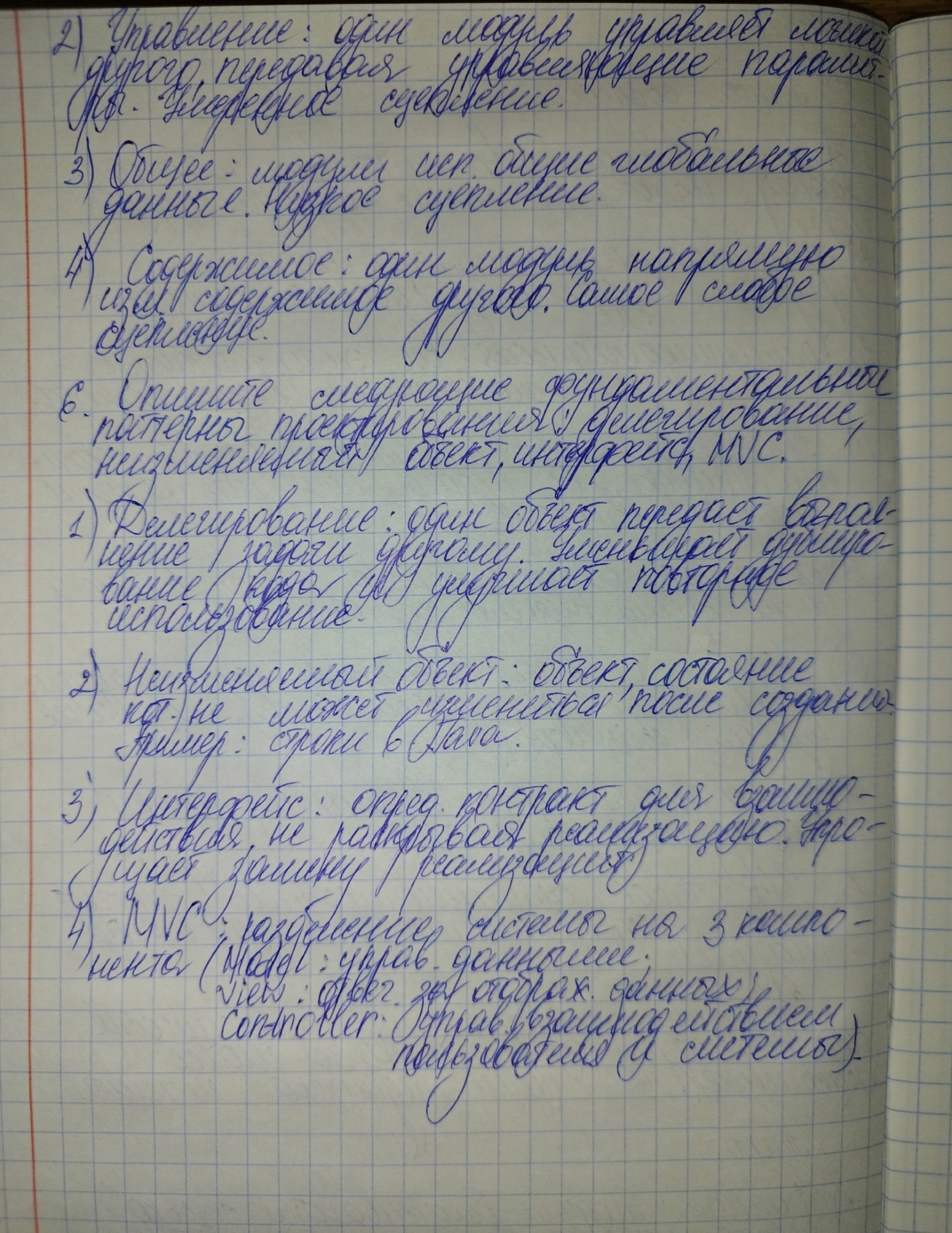
Контрольные вопросы:

* 1. В чем отличие предварительного проектирования от детального?
  2. Перечислите и кратко опишите архитектурные системные паттерны.
  3. Перечислите и кратко опишите паттерны управления.
  4. Что такое «связность модуля»? Перечислите и кратко опишите типы связности модуля с указанием значения силы связности для каждого типа.
  5. Что такое «сцепление модулей»? Перечислите и кратко опишите типы сцепления модулей.
  6. Опишите следующие фундаментальные паттерны проектирования: делегирование, неизменяемый объект, интерфейс, MVC.







Вывод: научилась проектировать простейшие системы и составлять документацию по проектированию программного продукта.

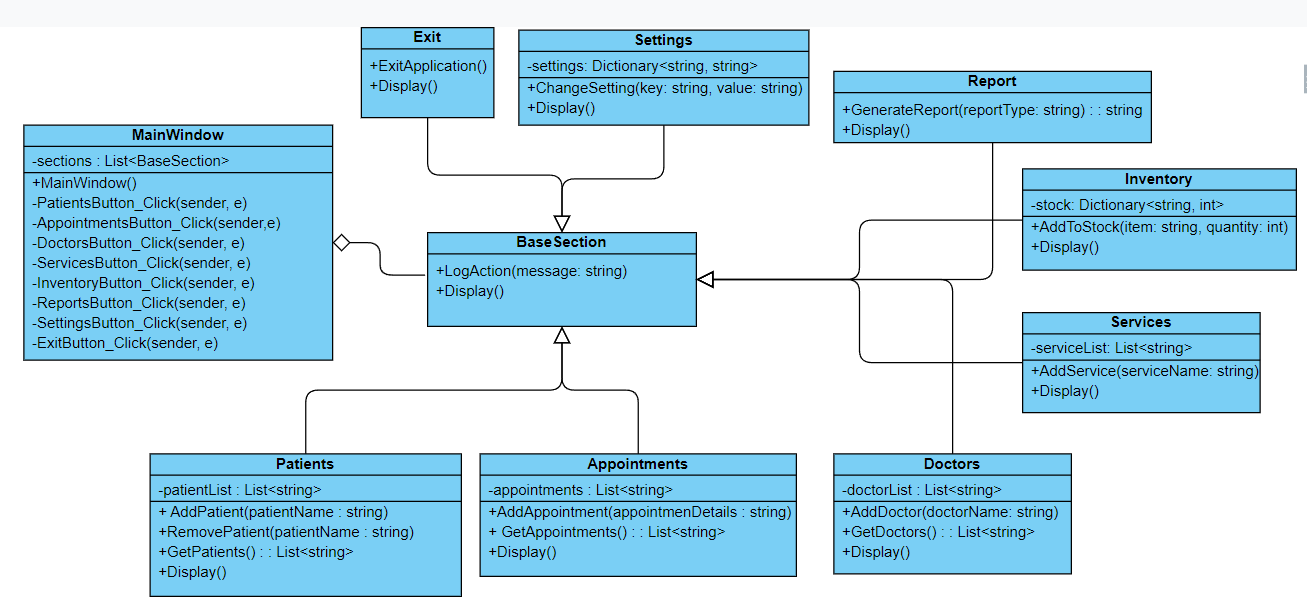
1. **Общая архитектура приложения:**

Рис.1 Архитектура приложения

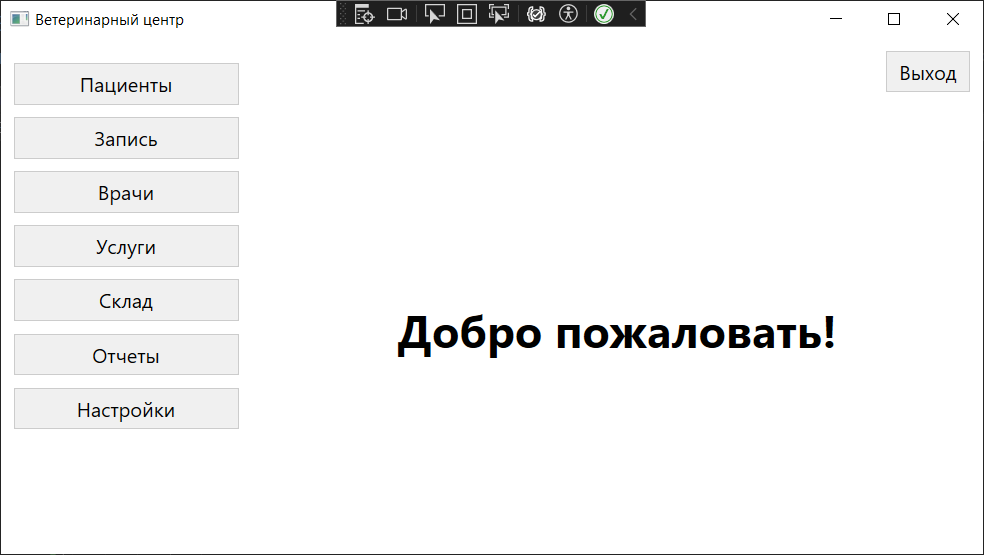
1. **Макет интерфейса программного продукта:**

Рис.2 Макет интерфейса программного продукта.

1. **Software Design Document (SDD)**

**Описание особенности системы ветеринарной клиники**

**Версия:** 1.0  
**Дата:** 18.12.2024

### ****СОДЕРЖАНИЕ****

1. Примечания
2. Структура проекта
3. Особенности реализации и использования классов
4. Описание Control Flow основных частей проекта
5. Форматы данных
6. **ПРИМЕЧАНИЯ**

* Код оформлен с использованием стандартов C# и WPF.
* Архитектура приложения основана на наследовании от абстрактного класса BaseSection, что упрощает добавление новых разделов.
* Метод SetActiveSection<T>() использует общий подход для работы с разными разделами, позволяя динамически переключать их.
* Именование переменных и методов соответствует стандартам C#, улучшая читаемость кода.
* Возможно улучшение с использованием интерфейсов для разделов и оптимизация обработки событий.
* Планируется расширение функционала, включая новые разделы и улучшенную обработку ошибок.

1. **СТРУКТУРА ПРОЕКТА**

**Логика приложения (классы):**

* BaseSection — абстрактный класс, от которого наследуются все разделы.
* MainWindow — главное окно, управляющее переключением между разделами.
* Patients, Appointments, Doctors, Services, Inventory, Reports, Settings, Exit — классы для управления данными и логикой каждого раздела.

**GUI-элементы (классы):**

* MainWindow — основное окно приложения, которое управляет отображением разделов и обработкой событий.
* Grid-классы (PatientsGrid, AppointmentsGrid и т.д.) — контейнеры для отображения данных каждого раздела.
* Button-классы (PatientsButton, AppointmentsButton и т.д.) — кнопки для навигации по разделам.
* ExitButton — кнопка для выхода из приложения.
* ComboBox, TextBox, ListBox — элементы управления для ввода и отображения данных.

**Служебные файлы приложения:**

* App.xaml и MainWindow.xaml — файлы конфигурации интерфейса.
* App.xaml.cs и MainWindow.xaml.cs — файлы логики приложения.

1. **ОСОБЕННОСТИ РЕАЛИЗАЦИИ И ИСПОЛЬЗОВАНИЯ КЛАССОВ**

**1. BaseSection (абстрактный класс)**

**Описание:**

BaseSection является абстрактным классом, от которого наследуются все другие разделы приложения. Этот класс реализует общую функциональность для всех разделов, такую как логирование действий с помощью метода LogAction() и предоставляет абстрактный метод Display(), который должен быть реализован в каждом разделе.

**Методы:**

* LogAction(string message) — метод для логирования действий в консоль.
* Display() — абстрактный метод для отображения данных соответствующего раздела (реализуется в наследующих классах).

**2. MainWindow (главное окно приложения)**

**Описание:**

MainWindow управляет всеми разделами приложения, а также переключает видимость разделов и обрабатывает события кнопок. Он содержит коллекцию разделов, таких как Пациенты, Записи, Врачи и др., и динамически отображает данные в зависимости от выбранного раздела.

**Методы:**

* SetActiveSection<T>() — позволяет активировать нужный раздел и отобразить его данные.
* Обработчики кнопок для каждого раздела (например, PatientsButton\_Click, AppointmentsButton\_Click) переключают активный раздел.

**3. Классы разделов**

Каждый из классов, таких как Patients, Appointments, Doctors, Services, Inventory, Reports, Settings, Exit, наследует BaseSection и реализует метод Display() для отображения информации в соответствующем разделе.

**Пример использования**:

* Patients: Управление пациентами.
* Appointments: Управление записями.
* Doctors: Управление врачами.
* Services: Управление услугами клиники.
* Inventory: Управление складом.
* Reports: Генерация отчетов.
* Settings: Управление настройками.
* Exit: Выход из приложения.

### ****ОПИСАНИЕ CONTROL FLOW ОСНОВНЫХ ЧАСТЕЙ ПРОЕКТ****

1. **Управление разделами:**
2. Пользователь нажимает кнопку для выбранного раздела (например, «Пациенты», «Запись», «Врачи» и т.д.).

* Каждая кнопка активирует соответствующий раздел в интерфейсе.

1. Раздел отображается в интерфейсе.

* После нажатия кнопки вызывается метод SetActiveSection<T>(), который активирует раздел и отображает его данные.

1. Пользователь взаимодействует с данным разделом (например, добавляет пациента, запись и др.).

* В зависимости от выбранного раздела, пользователь может добавлять, удалять или изменять данные.

1. Результат отображается в соответствующем разделе.

* Измененные или добавленные данные отображаются в интерфейсе для взаимодействия с пользователем.

1. **Работа с кнопкой «Выход»:**

1. Пользователь нажимает кнопку «Выход».

* При нажатии на кнопку вызывается метод ExitApplication().

2. Приложение закрывается.

* Приложение завершает свою работу, вызывая метод Application.Current.Shutdown().

1. **Обработка ошибок:**

* Некорректные действия при взаимодействии с разделами:
  + Если в разделе происходит ошибка (например, добавление записи с некорректными данными), будет выведено сообщение о проблеме в консоли или на экране.
* Ошибки при выходе из приложения:
  + Если происходит ошибка при завершении работы приложения, отображается соответствующее сообщение.

1. **ФОРМАТЫ ДАННЫХ**

**1. Структура разделов**

Каждый раздел имеет свою структуру данных для хранения информации и работы с ней:

* **Пациенты**:
  + Список пациентов (List<string> patientList), где хранятся имена пациентов.
  + Могут быть дополнительные данные, такие как возраст, диагноз и дата регистрации.
* **Записи**:
  + Список записей (List<string> appointments), где хранятся детали записи на прием (например, информация о пациенте, враче, времени приема).
* **Врачи**:
  + Список врачей (List<string> doctorList), где хранятся имена врачей.
* **Услуги**:
  + Список услуг (List<string> serviceList), где хранятся названия предоставляемых услуг.
* **Склад**:
  + Словарь товаров на складе (Dictionary<string, int> stock), где ключом является название товара, а значением — его количество.
* **Отчеты**:
  + Строки отчетов (например, результат работы с GenerateReport(string reportType)).
* **Настройки**:
  + Словарь настроек (Dictionary<string, string> settings), где ключом является название настройки, а значением — текущее значение.

**2. Структура данных для работы с разделами**

В каждом из разделов используется структура данных для хранения информации:

* Пациенты: Список строк, где каждая строка — это имя пациента.
* Записи: Список строк, где каждая строка — это запись с деталями (например, дата и пациент).
* Врачи: Список строк с именами врачей.
* Склад: Словарь, где ключ — это название товара, а значение — количество.
* Отчеты: Строка для хранения информации о сгенерированном отчете.
* Настройки: Словарь, где ключ — это название настройки, а значение — ее текущее значение.

**3. Обработка ошибок**

Для обработки ошибок могут использоваться различные механизмы:

* Если данные введены неверно (например, при добавлении пациента или записи), выводится сообщение об ошибке.
* В случае с некорректным ключом (например, для настроек или разделов) ввод блокируется или появляется предупреждение.

**Программный код:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

namespace VeterinaryClinic

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

private List<BaseSection> sections;

public MainWindow()

{

InitializeComponent();

// Инициализация всех разделов

sections = new List<BaseSection>

{

new Patients(),

new Appointments(),

new Doctors(),

new Services(),

new Inventory(),

new Reports(),

new Settings(),

new Exit()

};

}

// Обработчик кнопки "Пациенты"

private void PatientsButton\_Click(object sender, RoutedEventArgs e)

{

SetActiveSection<Patients>();

}

// Обработчик кнопки "Запись"

private void AppointmentsButton\_Click(object sender, RoutedEventArgs e)

{

SetActiveSection<Appointments>();

}

// Обработчик кнопки "Врачи"

private void DoctorsButton\_Click(object sender, RoutedEventArgs e)

{

SetActiveSection<Doctors>();

}

// Обработчик кнопки "Услуги"

private void ServicesButton\_Click(object sender, RoutedEventArgs e)

{

SetActiveSection<Services>();

}

// Обработчик кнопки "Склад"

private void InventoryButton\_Click(object sender, RoutedEventArgs e)

{

SetActiveSection<Inventory>();

}

// Обработчик кнопки "Отчеты"

private void ReportsButton\_Click(object sender, RoutedEventArgs e)

{

SetActiveSection<Reports>();

}

// Обработчик кнопки "Настройки"

private void SettingsButton\_Click(object sender, RoutedEventArgs e)

{

SetActiveSection<Settings>();

}

// Обработчик кнопки "Выход"

private void ExitButton\_Click(object sender, RoutedEventArgs e)

{

var exitSection = sections.OfType<Exit>().FirstOrDefault();

exitSection?.ExitApplication();

}

// Метод для активации раздела

private void SetActiveSection<T>() where T : BaseSection

{

foreach (var section in sections)

{

if (section is T)

{

section.Display();

}

}

}

}

// Базовый класс для всех разделов

public abstract class BaseSection

{

protected void LogAction(string message)

{

Console.WriteLine($"[Log]: {message}");

}

public abstract void Display();

}

// Класс для работы с пациентами

public class Patients : BaseSection

{

private List<string> patientList = new List<string>();

public void AddPatient(string patientName)

{

patientList.Add(patientName);

LogAction($"Пациент '{patientName}' добавлен.");

}

public void RemovePatient(string patientName)

{

patientList.Remove(patientName);

LogAction($"Пациент '{patientName}' удален.");

}

public List<string> GetPatients()

{

return patientList;

}

public override void Display()

{

Console.WriteLine("Отображение раздела 'Пациенты'.");

}

}

// Класс для работы с записями

public class Appointments : BaseSection

{

private List<string> appointments = new List<string>();

public void AddAppointment(string appointmentDetails)

{

appointments.Add(appointmentDetails);

LogAction($"Добавлена запись: {appointmentDetails}");

}

public List<string> GetAppointments()

{

return appointments;

}

public override void Display()

{

Console.WriteLine("Отображение раздела 'Записи'.");

}

}

// Класс для работы с врачами

public class Doctors : BaseSection

{

private List<string> doctorList = new List<string>();

public void AddDoctor(string doctorName)

{

doctorList.Add(doctorName);

LogAction($"Врач '{doctorName}' добавлен.");

}

public List<string> GetDoctors()

{

return doctorList;

}

public override void Display()

{

Console.WriteLine("Отображение раздела 'Врачи'.");

}

}

// Класс для работы с услугами

public class Services : BaseSection

{

private List<string> serviceList = new List<string>();

public void AddService(string serviceName)

{

serviceList.Add(serviceName);

LogAction($"Услуга '{serviceName}' добавлена.");

}

public override void Display()

{

Console.WriteLine("Отображение раздела 'Услуги'.");

}

}

// Класс для работы со складом

public class Inventory : BaseSection

{

private Dictionary<string, int> stock = new Dictionary<string, int>();

public void AddToStock(string item, int quantity)

{

if (stock.ContainsKey(item))

stock[item] += quantity;

else

stock[item] = quantity;

LogAction($"Добавлено на склад: {item} ({quantity}).");

}

public override void Display()

{

Console.WriteLine("Отображение раздела 'Склад'.");

}

}

// Класс для работы с отчетами

public class Reports : BaseSection

{

public string GenerateReport(string reportType)

{

LogAction($"Отчет '{reportType}' создан.");

return $"Отчет по {reportType}";

}

public override void Display()

{

Console.WriteLine("Отображение раздела 'Отчеты'.");

}

}

// Класс для работы с настройками

public class Settings : BaseSection

{

private Dictionary<string, string> settings = new Dictionary<string, string>();

public void ChangeSetting(string key, string value)

{

settings[key] = value;

LogAction($"Изменена настройка '{key}' на '{value}'.");

}

public override void Display()

{

Console.WriteLine("Отображение раздела 'Настройки'.");

}

}

// Класс для выхода из приложения

public class Exit : BaseSection

{

public void ExitApplication()

{

LogAction("Выход из приложения.");

Application.Current.Shutdown();

}

public override void Display()

{

Console.WriteLine("Выход из приложения.");

}

}

}