

**ЛАБОРАТОРНАЯ РАБОТА №6.
МНОГОПОТОЧНОСТЬ**

СОДЕРЖАНИЕ

Цель.....	2
Задание	2
Проектирование.....	2
Реализация.....	5
Контрольный пример	11
Требования.....	12
Порядок сдачи базовой части	13
Контрольные вопросы к базовой части	13
Усложненная лабораторная (необязательно)	13
Порядок сдачи усложненной части.....	14
Контрольные вопросы к усложненной части.....	14
Варианты	14

Цель

Изучить основы асинхронности и многопоточности.

Задание

1. Создать ветку от ветки пятой лабораторной.
2. Добавить исполнителя в приложение:
 - a. Добавить сущность «Исполнитель». По исполнителю необходимо хранить информацию: ФИО, пароль, стаж работы и квалификация.
 - b. В заказе фиксировать какой исполнитель выполнил заказ.
 - c. В RestAPI-приложение добавить возможность авторизоваться исполнителю, получать список новых заказов, брать их в работу и отмечать их выполнение.
 - d. Реализовать функционал, имитирующий выполнение исполнителями заказов (перевод в статусы «в работу» с назначением исполнителя и в статус «Готово»)
3. Вылить полученный результат в созданную ветку. Убедитесь, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

Проектирование

Проект расширяется, новая команда будет заниматься разработкой приложения для сотрудников, чтобы они сами могли фиксировать выполняемые заказы. Что это будет за приложение, web или мобильное мы не знаем. Наша задача проста: расширить функционал RestAPI-приложения, чтобы новая команда могла слать к нему запросы для получения и обновления данных (рисунок 6.1).



Рисунок 6.1 – Положение нового сервиса в архитектуре проекта

Сперва разберемся с алгоритмом обработки заказа исполнителем:

- Из списка новых заказов (в статусе «Принят») сотрудник выбирает заказ (переводит его в статус «Выполняется» и указывает, что он его делает).
- В течении какого-то времени он его делает.
- После выполнения заказа он отмечает это (переводит в статус «Готов»).
- Делает небольшой перерыв.
- Берет следующий заказ.
- Если по какой-то причине сотрудник не успевает сделать заказ до конца рабочего дня, на следующий день он начинает работу с выполнения этого заказа.

Перейдем к нашему проекту. Первым делом вводим новую сущность «Исполнитель». Тут та же схема, что и с «Клиентом». У «Исполнителя» 4 поля: ФИО, пароль, стаж работы и квалификация (ограничение, не должно быть 2-х исполнителей с одинаковым ФИО). В слое моделей объявим новый интерфейс модели-исполнителя, в слое контрактов объявим binding и view-модели, интерфейс бизнес-логики и интерфейс работы с данными, в слое бизнес-логики создать класс-реализацию интерфейса бизнес-логики, а в слое

хранения данных сделать 3 модели для хранения данных и 3 реализации интерфейса работы с данными.

Также вновь потребуется доработать сущность «Заказ». В модель добавить поле с идентификатором исполнителя. Причем, исполнитель будет проставляться, когда заказ будет переводится в статус «В работе», так что изначально он будет иметь значение null. Дополнить модели «Заказа» в слое контрактов, дополнить и доработать логику в моделях «Заказ» в слое хранения данных, а также добавить в метод `GetFilteredList` новое условие выборки заказов по статусу (отбирать новые заказы) и в метод `GetElement` условие для получения заказа исполнителя, который находится в данный момент в статусе «В работе», чтобы исполнитель мог его доделать.

Перейдем к desktop-приложению. Тут потребуется добавить новую форму для отображения списка исполнителей и форму для создания/редактирования исполнителя. Потребуется доработать логику вывода списка заказов на форму, чтобы отображалось ФИО исполнителя, выполнившего заказ. А с главной формы убрать кнопки смены статуса заказа «В работу» и «Готово». Чтобы иметь возможность с приложения менять статусы заказа и проверять корректность работы логики, сделаем логику-имитацию работы исполнителей. Для запуска ее работы на форме добавим пункт меню.

Далее RestAPI-приложение. Добавим новый контроллер, который будет позволять аутентифицироваться сотруднику, получить список новых заказов, взять заказ в работу и отметить о его готовности, а также получить заказ, который он не доделал в прошлый раз.

И последнее – разработка класса-имитации работы сотрудников. Для каждого сотрудника выполняется следующий алгоритм:

- выбор среди новых заказов заказа в работу, выполнение заказа;
- поиск по заказам, которые в статусе «Выполняются» и закреплены за сотрудником, выполнение заказа.

Так как под каждого сотрудника будет выделен отдельный поток исполнения, то важно учитывать, что сразу несколько работников могут пытаться взять заказ в работу, так что метод перевода заказа в работу следует ограничить так, чтобы только один поток мог его вызывать в единицу времени.

Реализация

Создадим новый интерфейс модели-исполнителя (листинг 6.1).

```
namespace AbstractShopDataModels.Models
{
    public interface IImplementerModel : IIid
    {
        string ImplementerFIO { get; }

        string Password { get; }

        int WorkExperience { get; }

        int Qualification { get; }
    }
}
```

Листинг 6.1 – Интерфейс ImplementerModel

Далее все также, что и для «Клиента». Потребуется создать binding и view-модели, интерфейс бизнес-логики и ее реализацию, интерфейс работы с данными и 3 реализации, 3 модели для хранения данных, формы для desktop-приложения (разработать самостоятельно). И внести доработки по сущности «Заказ» (разработать самостоятельно, не забыть про вызов метода перевода заказа в работу на однопоточный режим).

В проекте **AbstractShopRestApi** создадим новый контроллер и пропишем в нем методы, которые определили на этапе проектирования (листинг 6.2).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;
using AbstractShopDataModels.Enums;
using Microsoft.AspNetCore.Mvc;

namespace AbstractShopRestApi.Controllers
{
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class ImplementerController : Controller
    {
        private readonly ILogger _logger;
    }
}
```

```

private readonly IOrderLogic _order;

private readonly IImplementerLogic _logic;

public ImplementerController(IOrderLogic order, IImplementerLogic logic,
ILogger<ImplementerController> logger)
{
    _logger = logger;
    _order = order;
    _logic = logic;
}

[HttpGet]
public ImplementerViewModel? Login(string login, string password)
{
    try
    {
        return _logic.ReadElement(new ImplementerSearchModel
        {
            ImplementerFIO = login,
            Password = password
        });
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Ошибка авторизации сотрудника");
        throw;
    }
}

[HttpGet]
public List<OrderViewModel>? GetNewOrders()
{
    try
    {
        return _order.ReadList(new OrderSearchModel
        {
            Status = OrderStatus.Принят
        });
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Ошибка получения новых заказов");
        throw;
    }
}

[HttpGet]
public OrderViewModel? GetImplementerOrder(int implementerId)
{
    try
    {
        return _order.ReadElement(new OrderSearchModel
        {
            ImplementerId = implementerId
        });
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Ошибка получения текущего заказа
исполнителя");
        throw;
    }
}

```

```

        [HttpPost]
        public void TakeOrderInWork(OrderBindingModel model)
        {
            try
            {
                _order.TakeOrderInWork(model);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка перевода заказа с %Id в работу",
model.Id);
                throw;
            }
        }

        [HttpPost]
        public void FinishOrder(OrderBindingModel model)
        {
            try
            {
                _order.FinishOrder(model);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка отметки о готовности заказа с
%Id", model.Id);
                throw;
            }
        }
    }
}

```

Листинг 6.2 – Контроллер ImplementerController

Перейдем к имитации деятельности сотрудника. В контрактах зададим интерфейс для объявления общей логики работы имитации (листинг 6.3).

```

namespace AbstractShopContracts.BusinessLogicsContracts
{
    public interface IWorkProcess
    {
        /// <summary>
        /// Запуск работ
        /// </summary>
        void DoWork(IImplementerLogic implementerLogic, IOrderLogic orderLogic);
    }
}

```

Листинг 6.3 – Интерфейс IWorkProcess

В проекте **AbstractShopBusinessLogic** создадим класс-реализацию для этого интерфейса, в котором будем моделировать работы исполнителей (листинг 6.4).

```

using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;
using AbstractShopDataModels.Enums;
using Microsoft.Extensions.Logging;

namespace AbstractShopBusinessLogic.BusinessLogics

```

```

{
    public class WorkModeling : IWorkProcess
    {
        private readonly ILogger _logger;

        private readonly Random _rnd;

        private IOrderLogic? _orderLogic;

        public WorkModeling(ILogger<WorkModeling> logger)
        {
            _logger = logger;
            _rnd = new Random(1000);
        }

        public void DoWork(IImplementerLogic implementerLogic, IOrderLogic
orderLogic)
        {
            _orderLogic = orderLogic;
            var implementers = implementerLogic.ReadList(null);
            if (implementers == null)
            {
                _logger.LogWarning("DoWork. Implementers is null");
                return;
            }
            var orders = _orderLogic.ReadList(new OrderSearchModel { Status =
OrderStatus.Принят });
            if (orders == null || orders.Count == 0)
            {
                _logger.LogWarning("DoWork. Orders is null or empty");
                return;
            }
            _logger.LogDebug("DoWork for {Count} orders", orders.Count);
            foreach (var implementer in implementers)
            {
                Task.Run(() => WorkerWorkAsync(implementer, orders));
            }
        }

        /// <summary>
        /// Имитация работы исполнителя
        /// </summary>
        /// <param name="implementer"></param>
        /// <param name="orders"></param>
        private async Task WorkerWorkAsync(ImplementerViewModel implementer,
List<OrderViewModel> orders)
        {
            if (_orderLogic == null || implementer == null)
            {
                return;
            }
            await RunOrderInWork(implementer);

            await Task.Run(() =>
            {
                foreach (var order in orders)
                {
                    try
                    {
                        _logger.LogDebug("DoWork. Worker {Id} try get order
{Order}", implementer.Id, order.Id);
                        // пытаемся назначить заказ на исполнителя
                        _orderLogic.TakeOrderInWork(new OrderBindingModel
                        {
                            Id = order.Id,

```



```

        ImplementerId = implementer.Id
    });
    // делаем работу
    Thread.Sleep(implementer.WorkExperience * _rnd.Next(100,
1000) * order.Count);
    _logger.LogDebug("DoWork. Worker {Id} finish order
{Order}", implementer.Id, order.Id);
    _orderLogic.FinishOrder(new OrderBindingModel
    {
        Id = order.Id
    });
    }
    // кто-то мог уже перехватить заказ, игнорируем ошибку
    catch (InvalidOperationException ex)
    {
        _logger.LogWarning(ex, "Error try get work");
    }
    // заканчиваем выполнение имитации в случае иной ошибки
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error while do work");
        throw;
    }
    // отдыхаем
    Thread.Sleep(implementer.Qualification * _rnd.Next(10, 100));
    }
    });
}

/// <summary>
/// Ищем заказ, которые уже в работе (вдруг исполнителя прервали)
/// </summary>
/// <param name="implementer"></param>
/// <returns></returns>
private async Task RunOrderInWork(ImplementerViewModel implementer)
{
    if (_orderLogic == null || implementer == null)
    {
        return;
    }
    try
    {
        var runOrder = await Task.Run(() => _orderLogic.ReadElement(new
OrderSearchModel
        {
            ImplementerId = implementer.Id,
            Status = OrderStatus.Выполняется
        }));
        if (runOrder == null)
        {
            return;
        }

        _logger.LogDebug("DoWork. Worker {Id} back to order {Order}",
implementer.Id, runOrder.Id);
        // доделываем работу
        Thread.Sleep(implementer.WorkExperience * _rnd.Next(100, 300) *
runOrder.Count);
        _logger.LogDebug("DoWork. Worker {Id} finish order {Order}",
implementer.Id, runOrder.Id);
        _orderLogic.FinishOrder(new OrderBindingModel
        {
            Id = runOrder.Id
        });
        // отдыхаем

```

```

        Thread.Sleep(implementer.Qualification * _rnd.Next(10, 100));
    }
    // заказа может не быть, просто игнорируем ошибку
    catch (InvalidOperationException ex)
    {
        _logger.LogWarning(ex, "Error try get work");
    }
    // а может возникнуть иная ошибка, тогда просто заканчиваем
    выполнение имитации
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error while do work");
        throw;
    }
    }
}

```

Листинг 6.4 – Класс WorkModeling

Для работы нам понадобятся 2 класса-логики, `ImplementerLogic` (получение списка исполнителей) и `OrderLogic` (получение заказов, смена статусов). В главном методе будет получаться список исполнителей, а также список принятых заказов. Далее для каждого исполнителя будет вызываться метод (асинхронно) в нем сперва будет получаться заказ, который исполнитель не доделал по каким-то причинам (прервали работу имитации) и доделывать его (такого заказа может и не быть). После завершения этого заказа исполнитель пойдет по принятым заказам, будет пытаться получить заказ из списка, взять заказ в работу и выполнять его. При этом, цикл `foreach` по исполнителям не будет ждать, пока выполнится метод `WorkerWorkAsync` для первого исполнителя, а продолжит свою работу (т.е. будет вызывать метод `WorkerWorkAsync` для других исполнителей). Таким образом будет вызвано сразу несколько методов `WorkerWorkAsync` для разных исполнителей и все они сразу смогут выполнять заказы. Внутри метода `WorkerWorkAsync` будет вызов двух задач (`Task`). Метод не будет продолжать работу, пока каждый из вызовов `Task` не завершится (за счет вызова через `await`). Так что один сотрудник не успеет взять сразу все заказы в работу, а будет постепенно выполнять заказы.

В проекте **AbstractShopView** в главной форме добавляем пункт меню «Запуск работ» и в логике прописываем вызов метода интерфейса `IWorkProcess` (листинг 6.5).

```

e) private void ЗапускРаботToolStripMenuItem_Click(object sender, EventArgs
    {
        _workProcess.DoWork((Program.ServiceProvider?.GetService(typeof(ImplementerLogic
    )) as ImplementerLogic)!, _orderLogic);
        MessageBox.Show("Процесс обработки запущен", "Сообщение",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
    }

```

Листинг 6.5 – Запуск имитации работы исполнителей

Контрольный пример

Запускаем desktop-приложение и создаем несколько заказов (рисунок 6.2).

The screenshot shows a window titled 'Абстрактный магазин' (Abstract Store) with a menu bar containing 'Справочники', 'Отчеты', and 'Запуск работ'. The main area contains a table with the following data:

Номер	Клиент	Изделие	Исполнитель	Количество	Сумма	Статус	Дата создания	Дата выполнения
1	Иванов И.И.	Изделие 1		2	200,00	Принят	13.03.2022 22:57	
2	Петров П.П.	Изделие 2		5	750,00	Принят	13.03.2022 22:57	
3	Петров П.П.	Изделие 1		4	400,00	Принят	13.03.2022 22:57	

On the right side of the window, there are three buttons: 'Создать заказ', 'Заказ выдан', and 'Обновить список'.

Рисунок 6.2 – Главная форма с созданными заказами

Создаем нескольких исполнителей (рисунок 6.3).

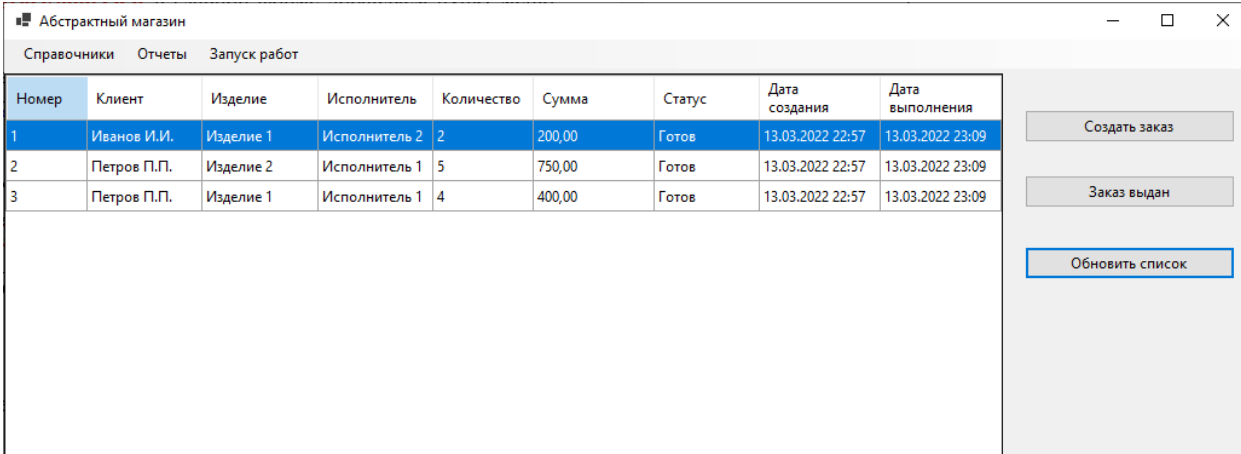
The screenshot shows a window titled 'Исполнители' (Executors) with a table containing the following data:

ФИО исполнителя	Пароль	Стаж работы	Квалификация
Исполнитель 1	пароль1	1	6
Исполнитель 2	пароль2	4	2

On the right side of the window, there are four buttons: 'Добавить', 'Изменить', 'Удалить', and 'Обновить'.

Рисунок 6.3 – Форма с исполнителями

Возвращаемся на главную форму и нажимаем пункт меню «Запуск работ». Обновляем список и убеждаемся, что сотрудники выполнили заказы (рисунок 6.4).



Номер	Клиент	Изделие	Исполнитель	Количество	Сумма	Статус	Дата создания	Дата выполнения
1	Иванов И.И.	Изделие 1	Исполнитель 2	2	200,00	Готов	13.03.2022 22:57	13.03.2022 23:09
2	Петров П.П.	Изделие 2	Исполнитель 1	5	750,00	Готов	13.03.2022 22:57	13.03.2022 23:09
3	Петров П.П.	Изделие 1	Исполнитель 1	4	400,00	Готов	13.03.2022 22:57	13.03.2022 23:09

Рисунок 6.4 – Главная форма с выполненными заказами

Требования

1. Название проектов должны **ОТЛИЧАТЬСЯ** от названия проектов, приведенных в примере и должны соответствовать логике вашего задания по варианту.
2. Название форм, классов, свойств классов должно соответствовать логике вашего задания по варианту.
3. **НЕ ИСПОЛЬЗОВАТЬ** в названии класса, связанного с изделием слово «Product» (во вариантах в скобках указано название класса для изделия)!!!
4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).
5. Сделать binding и view-модели для исполнителя, интерфейс бизнес-логики и интерфейс работы с данными.
6. Сделать реализацию ImplementerLogic (не должно быть 2-ч исполнителей с одинаковым ФИО).

7. Сделать реализацию моделей и интерфейса ImplementerStorage для хранилищ.
8. Добавить в сущность «Заказ» новое поле-идентификатор исполнителя и в метод GetFilteredList и GetElement новые варианты выборки.
9. В desktop-приложение добавить форму вывода списка исполнителей и форму создания/редактирования исполнителя.
10. Выводить ФИО исполнителя в заказах на главной форме.

Порядок сдачи базовой части

1. Предварительно создать 2-3 исполнителей и 5-8 заказов (5 если 2 исполнителя, 8 если 3 исполнителя; они должны оставаться в статусе «Принят»)
2. Запустить Desktop-проект, вызвать «Запуск работ», обновить список, показать, что заказы меняются
3. Ответить на вопрос преподавателя

Контрольные вопросы к базовой части

1. Как реализован однопоточный доступ к методу перевода заказа в работу?
2. Как сущность «Исполнитель» встроена в систему?
3. Как работает имитация деятельности сотрудника?

Усложненная лабораторная (необязательно)

1. В случае, если для выполнения заказов не хватает места в магазине, заказ должен переводиться в статус «Ожидание» и исполнитель должен будет переходить к следующему заказу.
2. Для исполнителя при имитации работы поменять алгоритм работы: сперва обрабатывается заказ со статусом «Ожидание», потом заказы

со статусом «Выполняется» (вдруг место освободилось) и только потом новые заказы.

Порядок сдачи усложненной части

1. Создать 6-7 заказов.
2. Запустить Desktop-проект, вызвать «Запуск работ», обновить список, показать, что заказы меняются (2-3 должны пройти, для остальных должно не хватить места).
3. Списать с магазинов изделия вызвать «Запуск работ», обновить список, показать, что заказы меняются (еще 2-3 должны пройти, для остальных должно не хватить места).
4. Ответить на вопрос преподавателя

Контрольные вопросы к усложненной части

1. Как реализован однопоточный доступ к методу перевода заказа в работу?
2. Как происходит выборка заказов со статусом «Ожидание»?
3. Как работает имитация деятельности сотрудника?

Варианты

1. Кондитерская. В качестве компонентов выступают различные виды шоколада и наполнители, типа орехов, изюма и т.п. Изделие – кондитерское изделие (pastry).
2. Автомастерская. В качестве компонентов выступают различные масла, смазки и т.п. Изделия – ремонт автомобиля (repair).
3. Моторный завод. В качестве компонентов выступают различные детали для производства двигателей. Изделия – двигатели (engine).
4. Суши-бар. В качестве компонентов выступают различные продукты для суши (рыба, водоросли, соусы). Изделия – суши (sushi).

5. Продажа компьютеров. В качестве компонентов выступают различные части для компьютеров (планки памяти, жесткие диски и т.п.). Изделия – компьютеры (computer).
6. Сборка мебели. В качестве компонентов выступают различные заготовки (ножки, спинки и т.п.). Изделия – мебель (furniture).
7. Рыбный завод. В качестве компонентов выступают различные виды рыб + дополнения к ним, типа соусов и т.п. Изделия – консервы (canned).
8. Установка ПО. В качестве компонентов выступают различное ПО. Изделия – пакеты установки, например, пакет установки офисных приложений, пакет разработчика и т.п. (package).
9. Ремонтные работы в помещении. В качестве компонентов выступают различные расходные материалы (клей, обои, краска, плитка, цемент и т.п.). Изделия – ремонтные работы в различных помещениях (repair).
10. Кузнечная мастерская. В качестве компонентов выступают различные болванки (заготовки), из которых изготавливаются подковы, кочерги и т.п. Изделия – кузнечные изделия (manufacture).
11. Пиццерия. В качестве компонентов выступают различные ингредиенты для пицц (тесто, соусы, паста и т.д.). Изделия – пиццы (pizza).
12. Завод ЖБИ. В качестве компонентов выступают различные виды бетона и металлоконструкций. Изделия – железобетонные изделия (reinforced).
13. Закусочная. В качестве компонентов выступают различные продукты для закусок (колбаса, сыр, хлеб и т.п.). Изделия – различные закуски (snack).
14. Пошив платьев. В качестве компонентов выступают различные ткани, нитки и т.п. Изделия – платья (dress).

15. Типография. В качестве компонентов выступают различные типы бумаг, тонер или чернила и т.п. Изделия – печатная продукция (листовки, брошюры, книги) (printed).
16. Автомобильный завод. В качестве компонентов выступают различные части для сборки автомобилей (кузов, двигатель, стекла и т.п.). Изделия – автомобили (car).
17. Юридическая фирма. В качестве компонентов выступают различные бланки для документов. Изделия – пакеты документов, например, для страховки или завещания (document).
18. Туристическая фирма. В качестве компонентов выступают различные условия поездки (отель проживания, туры в рамках поездок). Изделия – туристические путевки (travel).
19. Цветочная лавка. В качестве компонентов выступают различные цветы и украшения к ним. Изделия – цветочные композиции (flower).
20. Ювелирная лавка. В качестве компонентов выступают различные драгоценные камни и металлы. Изделия – драгоценности (jewel).
21. Авиастроительный завод. В качестве компонентов выступают различные части для сборки самолета (двигатели, крылья, фюзеляж и т.п.). Изделия – самолеты (plane).
22. Магазин подарков. В качестве компонентов выступают различные упаковочные материалы, ленты и подарки. Изделия – подарочные наборы (gift).
23. Система безопасности. В качестве компонентов выступают различные камеры, датчики и т.п. Изделия – базовые комплектации охраны, продвинутые, для предприятий, для частных и т.п. (secure).
24. Заказы еды. В качестве компонентов выступают различные блюда. Изделия – это наборы блюд (типа обеденный набор, или утренний набор, или набор для пикника) (dish).

25. Ремонт сантехники. В качестве компонентов выступают различные трубы, прокладки, смесители т.п. Изделия – замены смесителей, труб и т.п. (work).
26. Лавка с мороженым. В качестве компонентов выступают различные виды мороженого и добавки (орехи, шоколад и т.п.). Изделия – мороженое (icescream).
27. Судостроительный завод. В качестве компонентов выступают различные части для сборки судов (корпуса, двигатели и т.п.). Изделия – суда (ship).
28. Столярная мастерская. В качестве компонентов выступают различные деревянные заготовки. Изделия – деревянные игрушки, утварь и т.п. (wood).
29. Бар. В качестве компонентов выступают различные ингредиенты для коктейлей. Изделия – коктейли (cocktail).
30. Швейная фабрика. В качестве компонентов выступают различные заготовки для штор, покрывал и т.п. (textile).