ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)

КАФЕДРА АВТОМАТИЗАЦИИ ПРЕДПРИЯТИЙ СВЯЗИ

ПРЕДМЕТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ АВТОМАТИЗИРОВАННЫХ СИСТЕМ УПРАВЛЕНИЯ

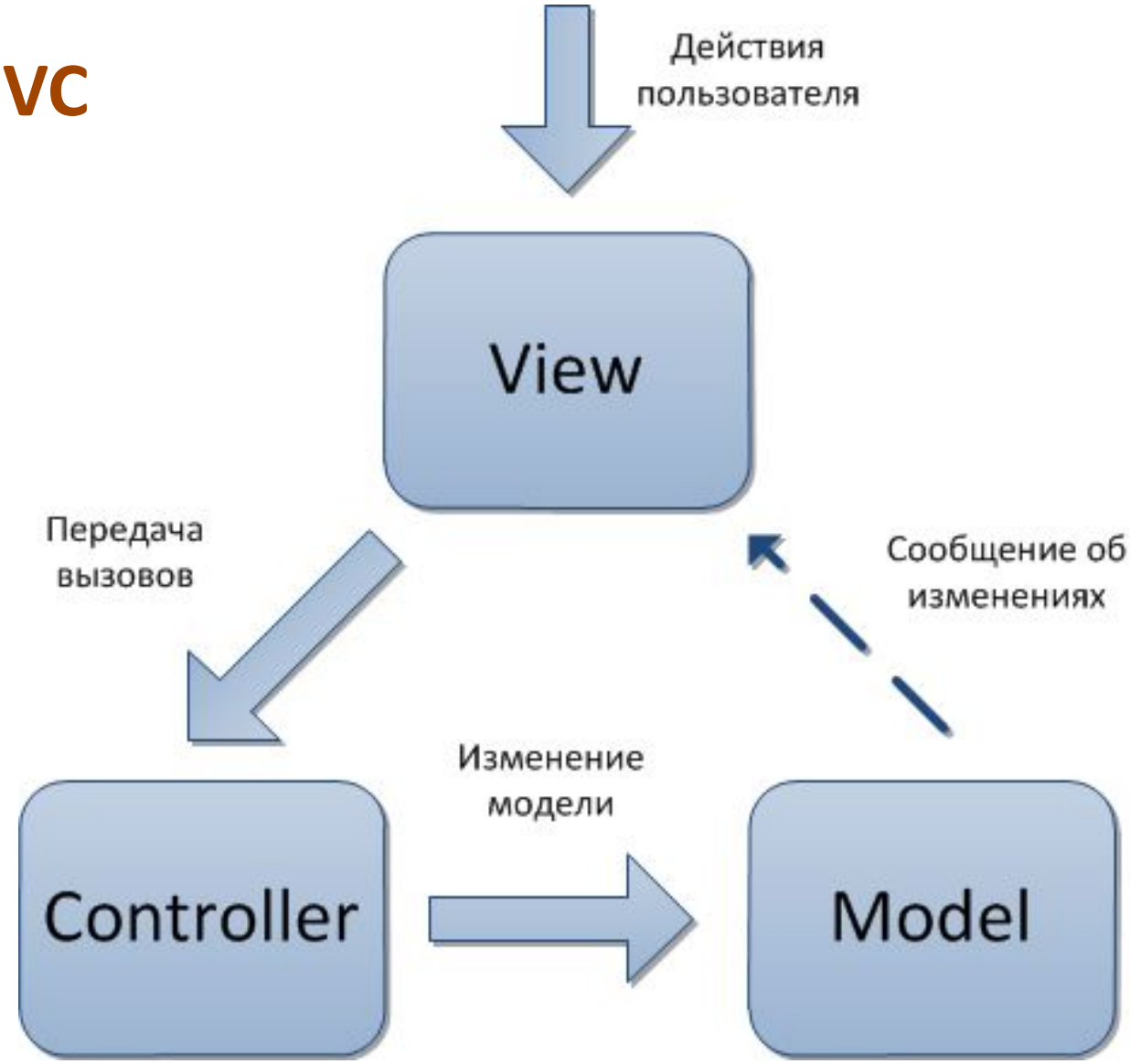
Раздел 2
Технологии DDD

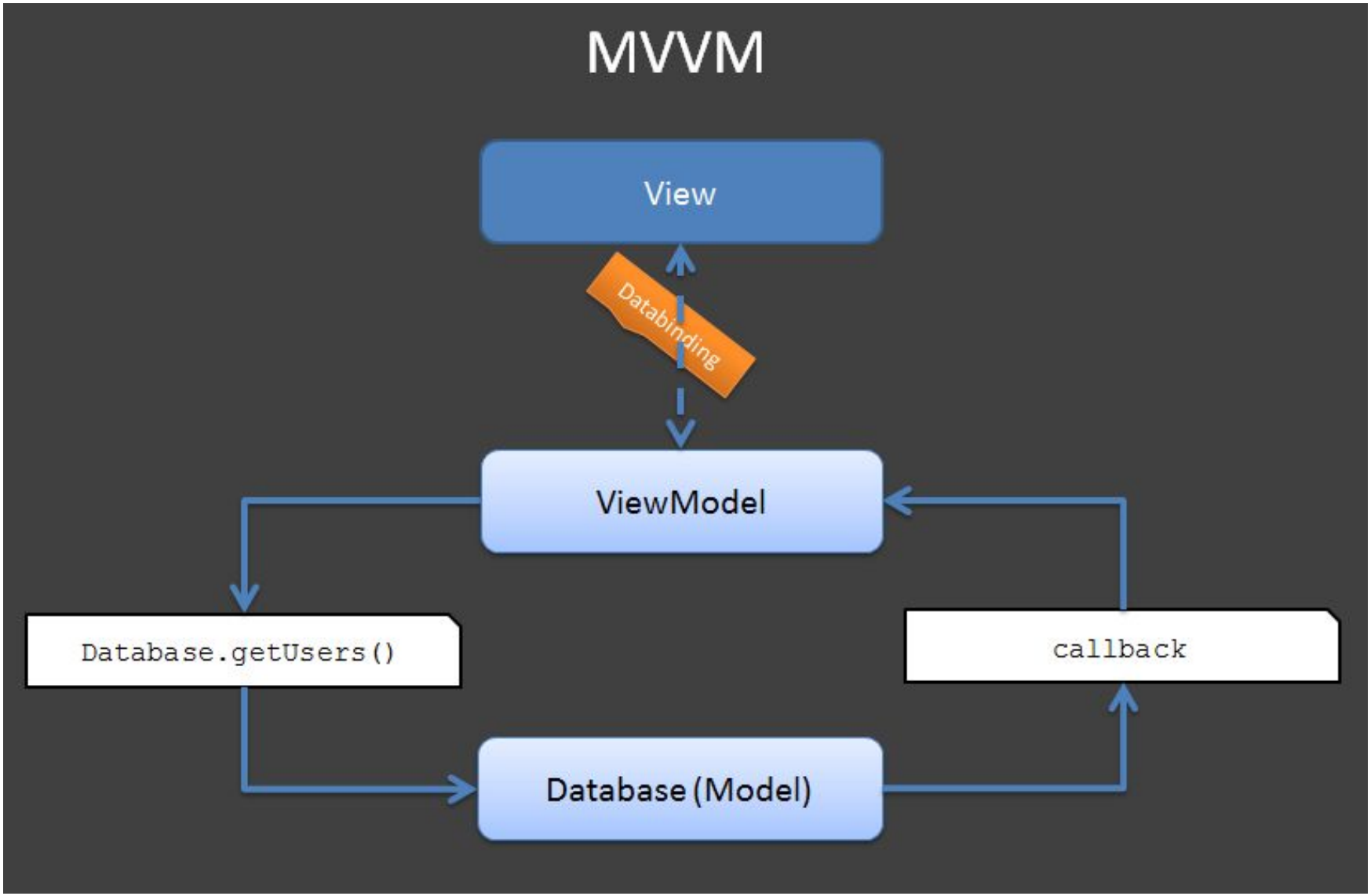
2019

РАЗДЕЛ 2
ТЕХНОЛОГИИ DDD

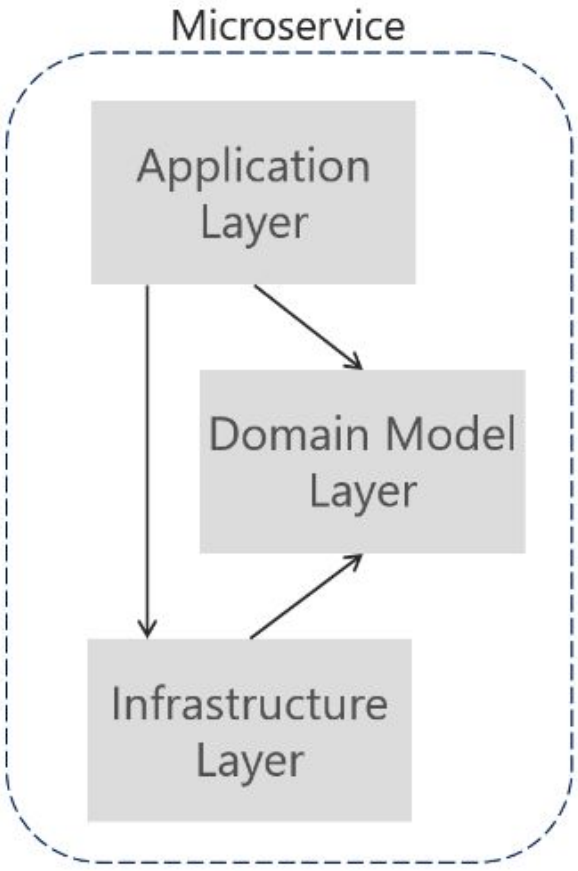
Архитектурные решения DDD

MVC


Действия пользователя
View
Передача вызовов
Сообщение об изменениях
Изменение модели
Controller
Model


MVVM
View
Databinding
ViewModel
Database.getUsers()
callback
Database (Model)

Dependencies between Layers in a Domain-Driven Design service

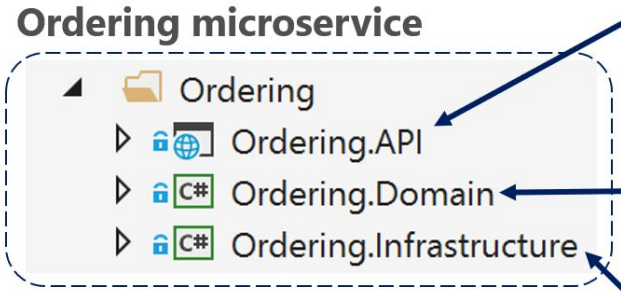


- Depends on the Domain-Model Layer so it can:
  - Use entity objects
  - Use Repository Interfaces/Contracts
- Depends on the Infrastructure Layer (thru DI) so it can:
  - Use Repository implementation classes, ideally through DI

- Ideally, it must NOT take dependency on any other layer
- It implements:
  - Domain Entities, Aggregate-Roots and Value-Objects
  - Repository Contracts/Interfaces (to be used in DI)

- Depends on the Domain-Model Layer so it can:
  - Use entity objects.
    - Like EF updating a database through mapped entities
- Direct dependency on infrastructure frameworks like EF Core or any other database, cache or infrastructure API

Layers in a Domain-Driven Design Microservice



Application layer

- ASP.NET Web API
- Network access to microservice
- API contracts/implementation
- Commands and command handlers
- Queries (when using an CQS approach)
  - Micro ORMs like Dapper

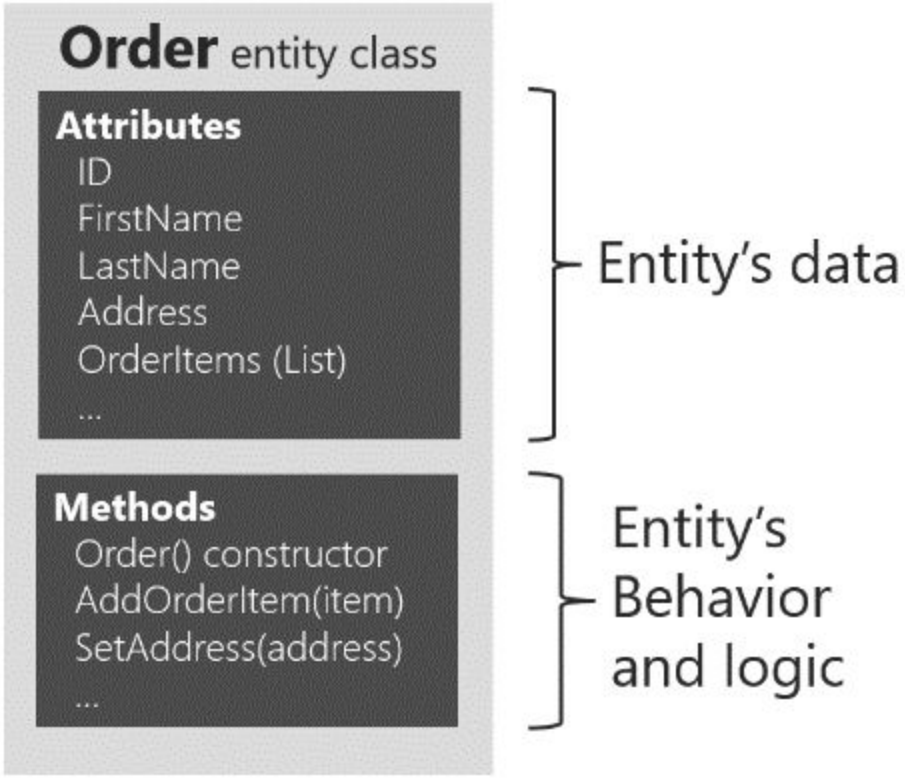
Domain model layer

- Domain entity model
- POCO entity classes (clean C# code)
- Domain entities with data + behavior
- DDD patterns:
  - Domain entity, aggregate
  - Aggregate root, value object
  - Repository contracts/interfaces

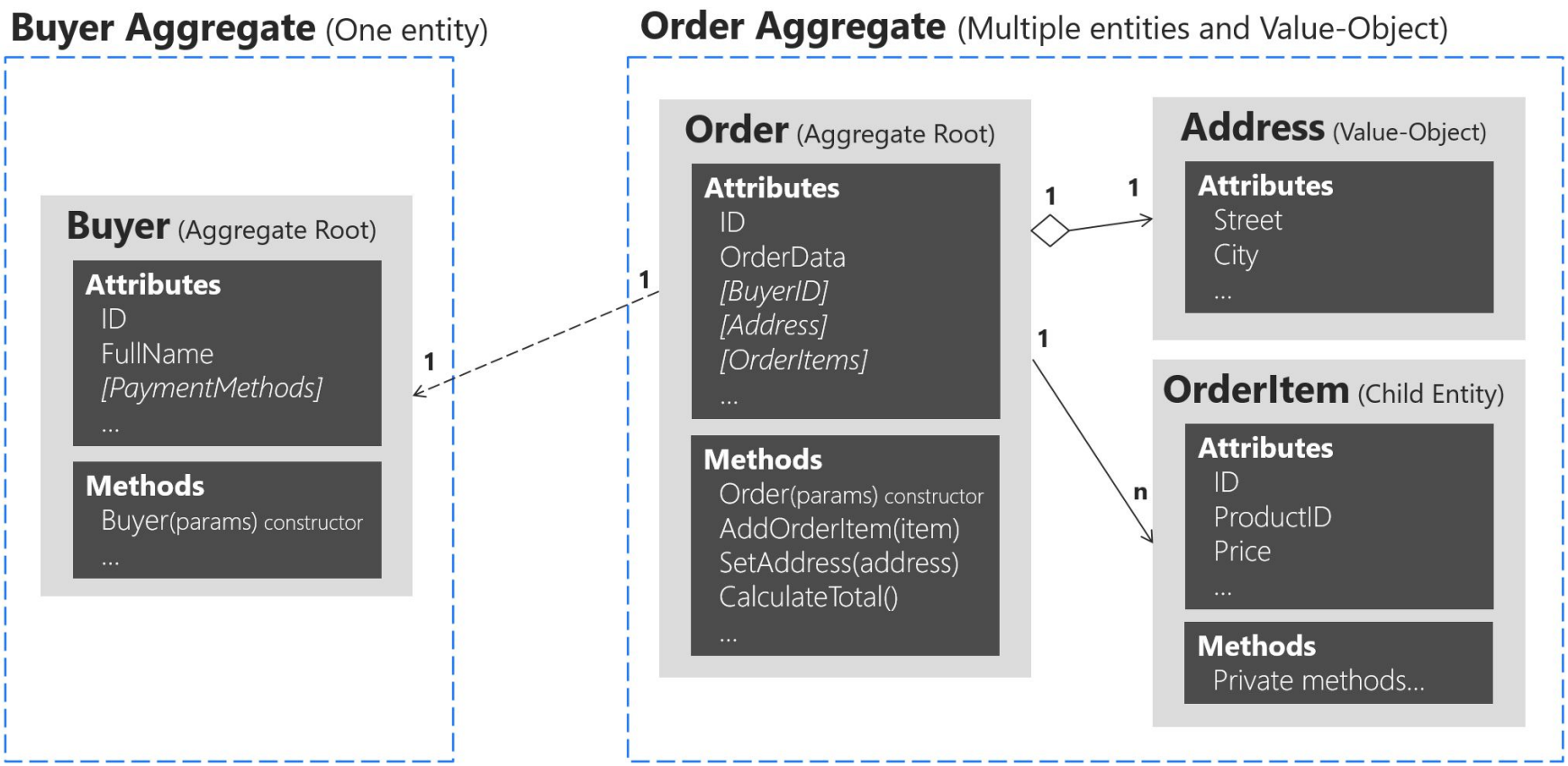
Infrastructure layer

- Data persistence infrastructure
  - Repository implementation
- Use of ORMs or data access API:
  - Entity Framework Core or any ORM
  - ADO.NET
  - Any NoSQL database API
- Other infrastructure implementation used from the application layer
  - Logging, cryptography, search engine, etc.

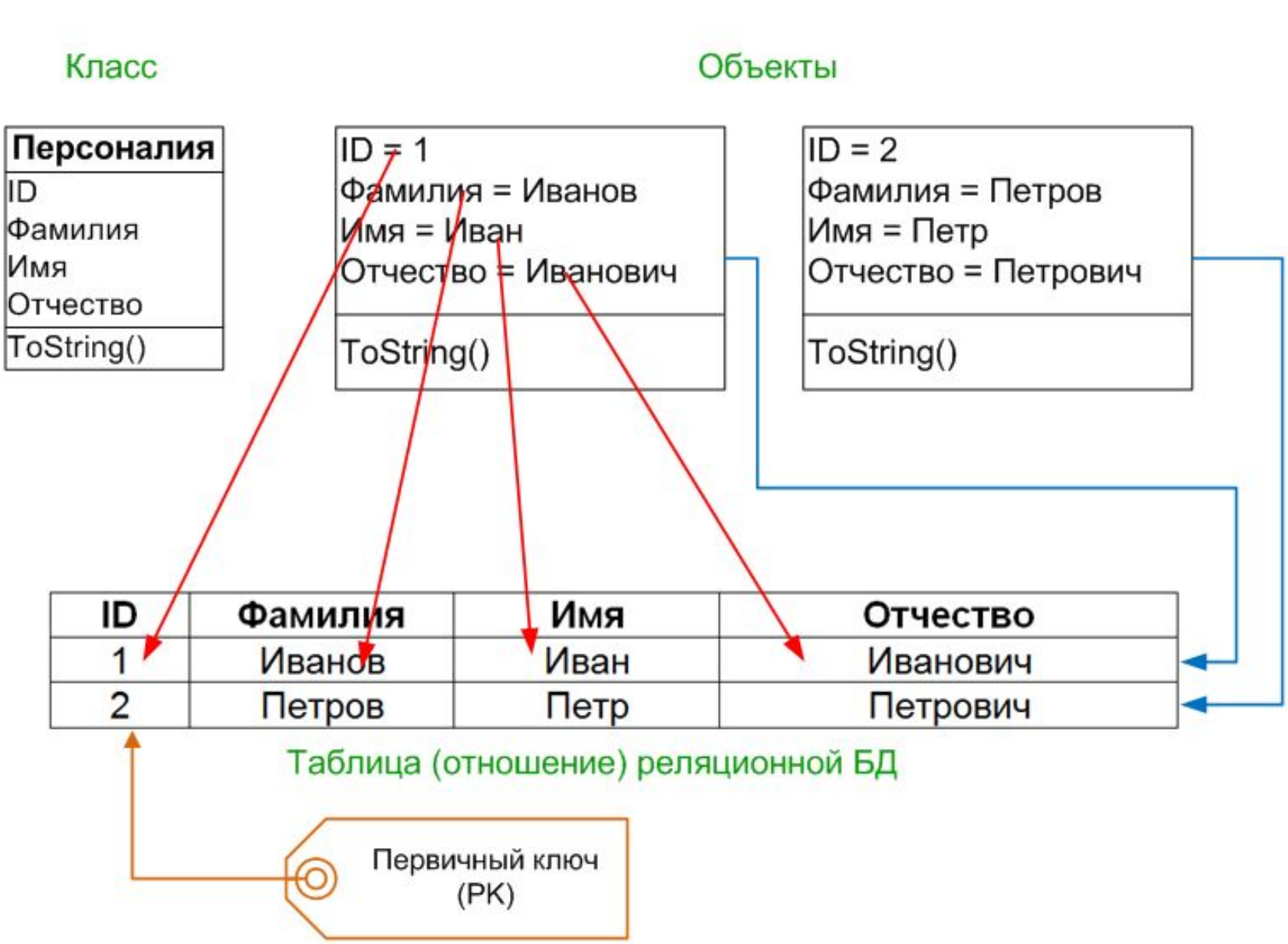
Domain Entity pattern

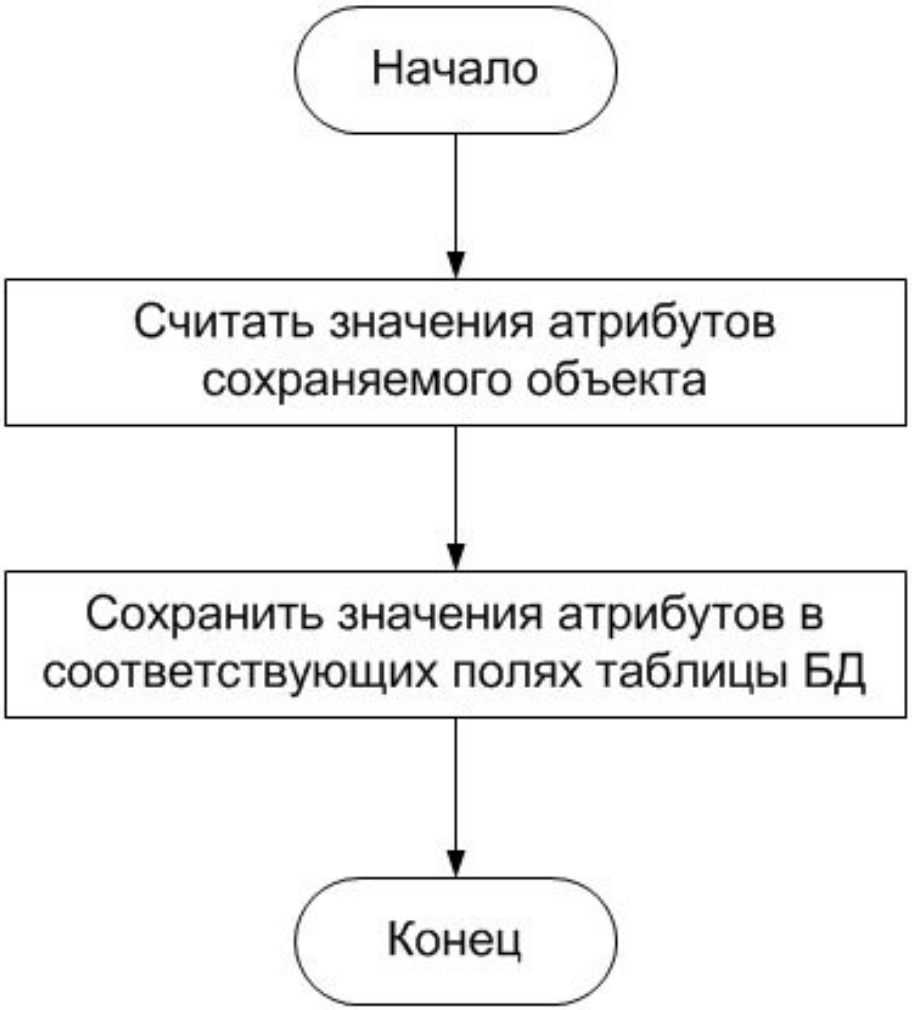


Aggregate pattern

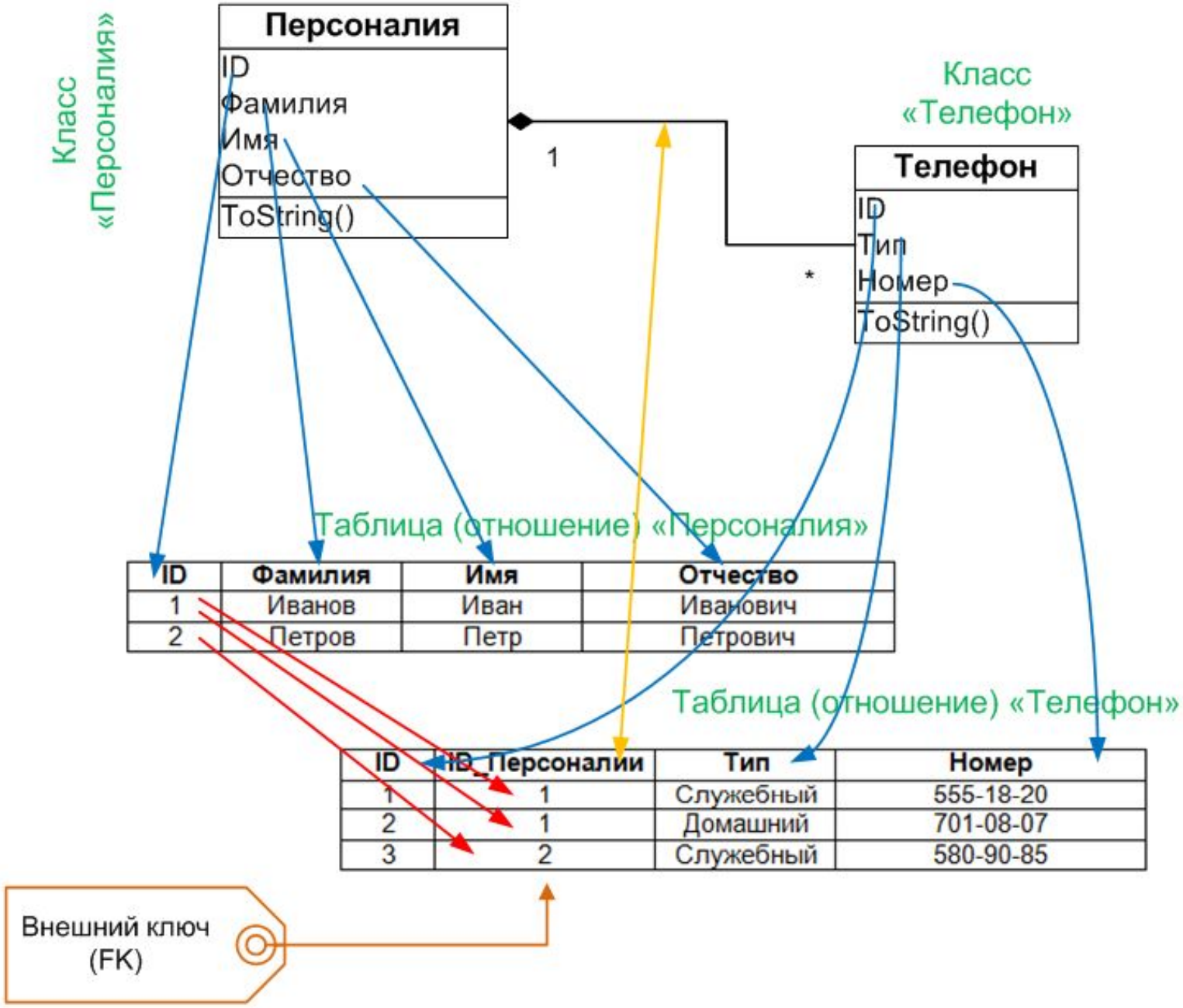


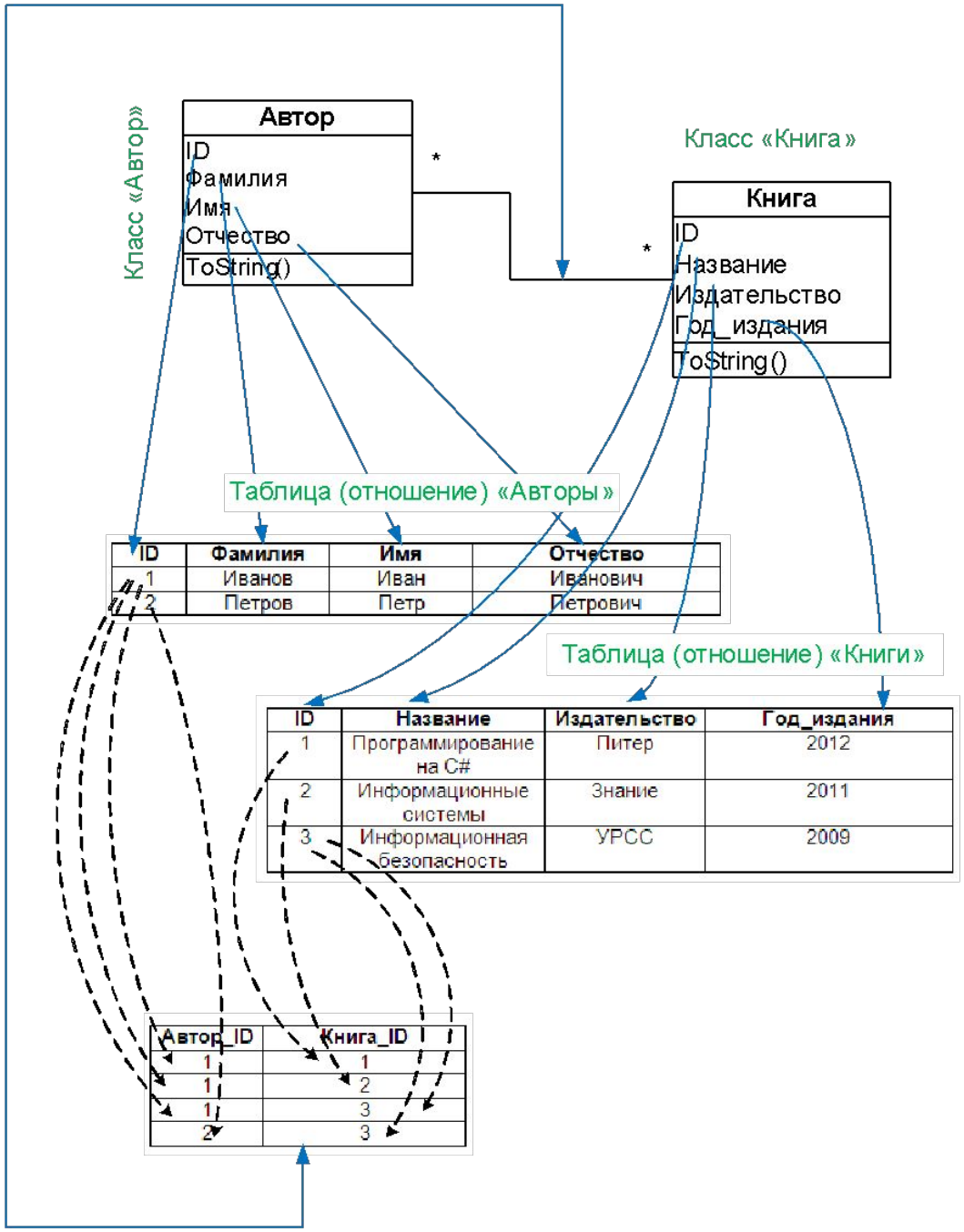
Объектно-реляционное преобразование


Класс
Объекты
Персоналия
ID
Фамилия
Имя
Отчество
ToString()
ID = 1
Фамилия = Иванов
Имя = Иван
Отчество = Иванович
ToString()
ID = 2
Фамилия = Петров
Имя = Петр
Отчество = Петрович
ToString()
ID
Фамилия
Имя
Отчество
1
Иванов
Иван
Иванович
2
Петров
Петр
Петрович
Таблица (отношение) реляционной БД
Первичный ключ
(PK)

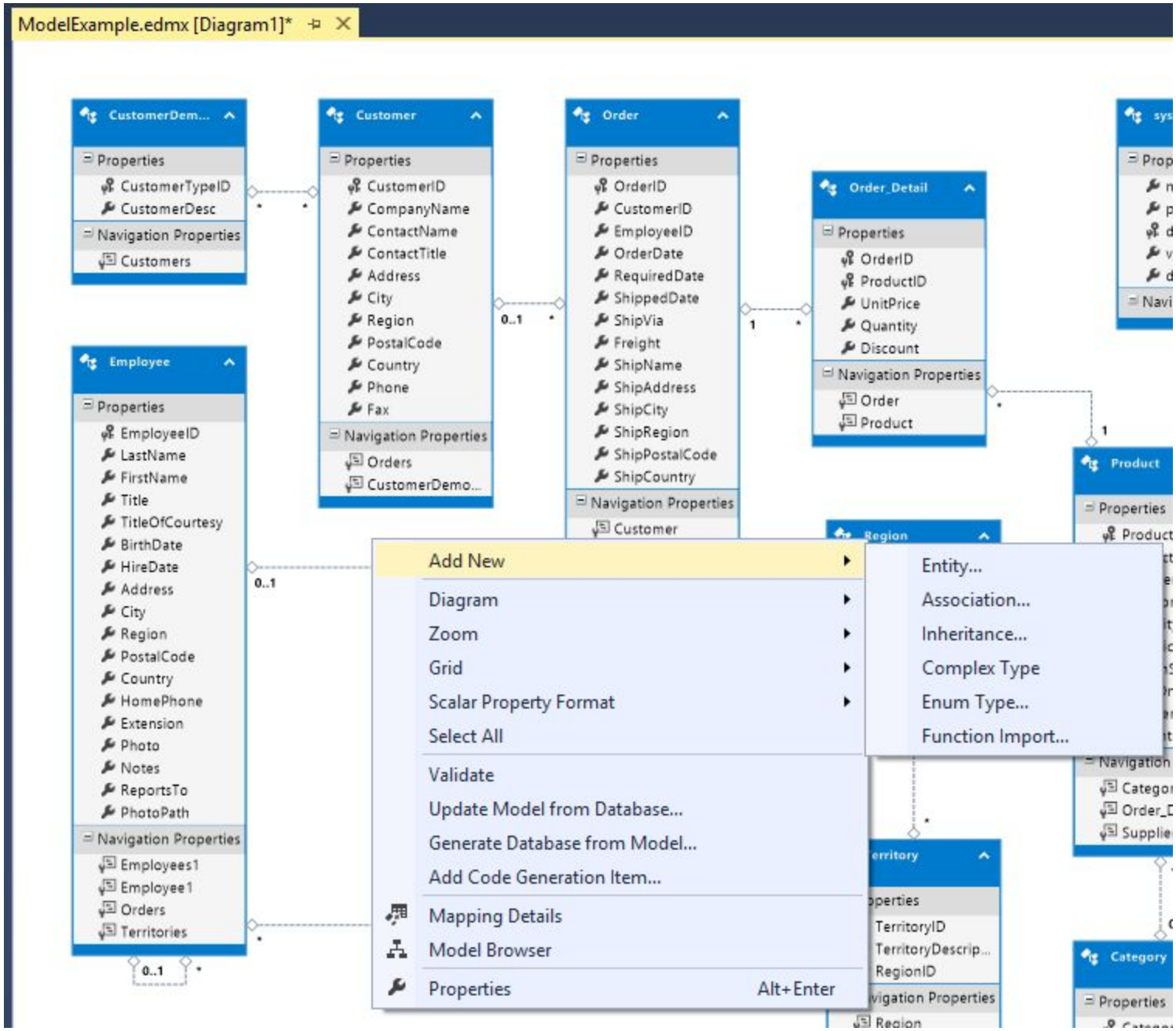

Начало
Считать значения атрибутов сохраняемого объекта
Сохранить значения атрибутов в соответствующих полях таблицы БД
Конец

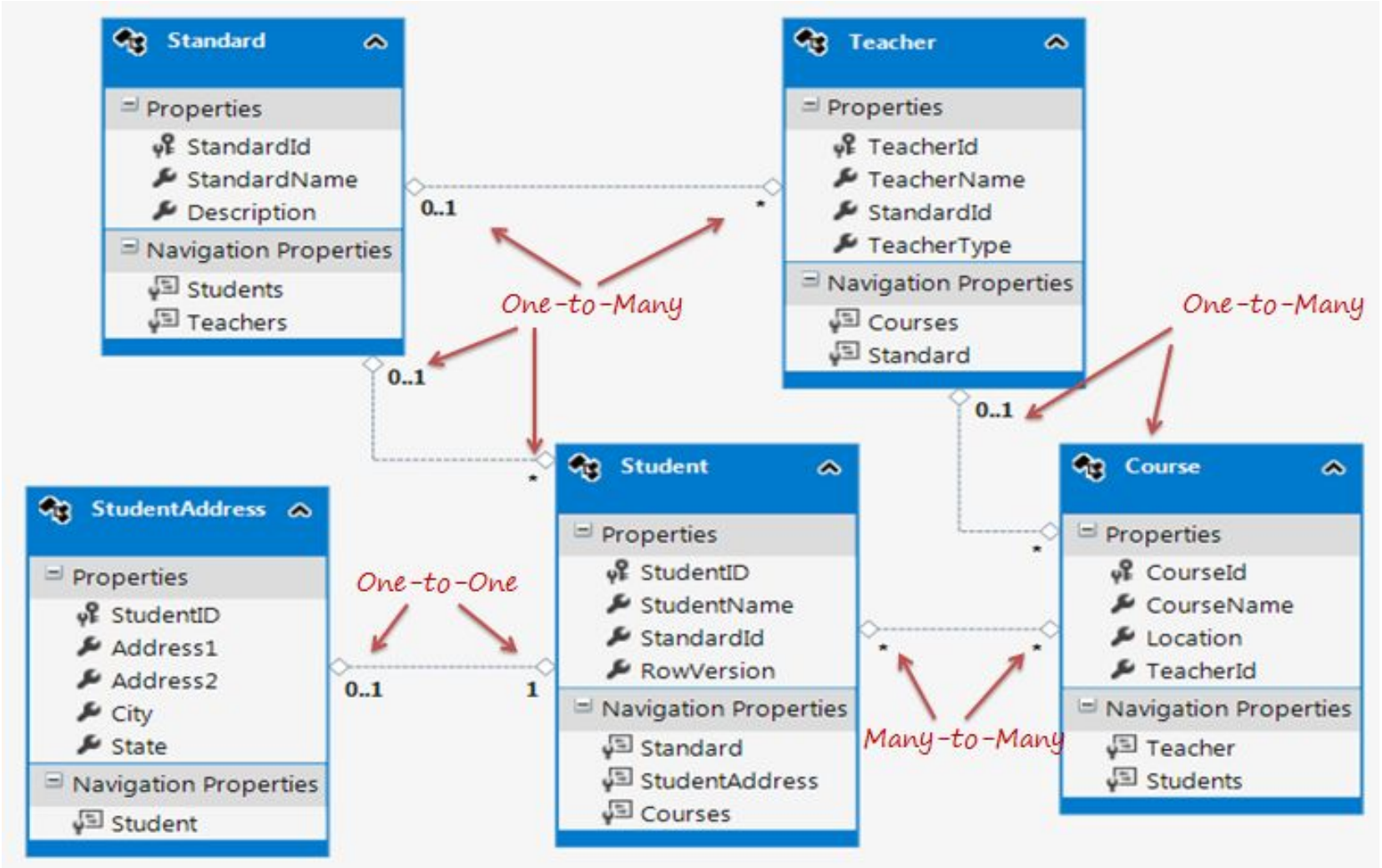

Начало
Создать объект
Извлечь запись БД по значению идентификатора восстанавливаемого объекта, в которой хранятся значения атрибутов
Считать значения полей, в которых хранятся значения атрибутов восстанавливаемого объекта
Присвоить атрибутам созданного объекта считанные значения
Конец


Класс «Персоналия»
Персоналия
ID
Фамилия
Имя
Отчество
ToString()
1
*
Класс «Телефон»
Телефон
ID
Тип
Номер
ToString()
Таблица (отношение) «Персоналия»
ID
Фамилия
Имя
Отчество
1
Иванов
Иван
Иванович
2
Петров
Петр
Петрович
Таблица (отношение) «Телефон»
ID
ID_Персоналии
Тип
Номер
1
1
Служебный
555-18-20
2
1
Домашний
701-08-07
3
2
Служебный
580-90-85
Внешний ключ (FK)


Класс «Автор»
Автор
ID
Фамилия
Имя
Отчество
ToString()
*
Класс «Книга»
Книга
ID
Название
Издательство
Год_издания
ToString()
*
Таблица (отношение) «Авторы»
ID
Фамилия
Имя
Отчество
1
Иванов
Иван
Иванович
2
Петров
Петр
Петрович
Таблица (отношение) «Книги»
ID
Название
Издательство
Год_издания
1
Программирование на C#
Питер
2012
2
Информационные системы
Знание
2011
3
Информационная безопасность
УРСС
2009
Автор_ID
Книга_ID
1
1
1
2
1
3
2
3

Entity Framework


ModelExample.edmx [Diagram1]*
CustomerDem...
Properties
CustomerTypeID
CustomerDesc
Navigation Properties
Customers
Customer
Properties
CustomerID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax
Navigation Properties
Orders
CustomerDemo...
Order
Properties
OrderID
CustomerID
EmployeeID
OrderDate
RequiredDate
ShippedDate
ShipVia
Freight
ShipName
ShipAddress
ShipCity
ShipRegion
ShipPostalCode
ShipCountry
Navigation Properties
Customer
Order_Detail
Properties
OrderID
ProductID
UnitPrice
Quantity
Discount
Navigation Properties
Order
Product
Product
Properties
Navigation
Employee
Properties
EmployeeID
LastName
FirstName
Title
TitleOfCourtesy
BirthDate
HireDate
Address
City
Region
PostalCode
Country
HomePhone
Extension
Photo
Notes
ReportsTo
PhotoPath
Navigation Properties
Employees1
Employee1
Orders
Territories
Region
Territory
TerritoryID
TerritoryDescrip...
RegionID
Category
Properties
0..1
1
*
Add New
Diagram
Zoom
Grid
Scalar Property Format
Select All
Validate
Update Model from Database...
Generate Database from Model...
Add Code Generation Item...
Mapping Details
Model Browser
Properties
Alt+Enter
Entity...
Association...
Inheritance...
Complex Type
Enum Type...
Function Import...


Standard
Properties
StandardId
StandardName
Description
Navigation Properties
Students
Teachers
Teacher
Properties
TeacherId
TeacherName
StandardId
TeacherType
Navigation Properties
Courses
Standard
0..1
*
One-to-Many
0..1
*
One-to-Many
0..1
*
StudentAddress
Properties
StudentID
Address1
Address2
City
State
Navigation Properties
Student
One-to-One
0..1
1
Student
Properties
StudentID
StudentName
StandardId
RowVersion
Navigation Properties
Standard
StudentAddress
Courses
* *
Many-to-Many
Course
Properties
CourseId
CourseName
Location
TeacherId
Navigation Properties
Teacher
Students

Назначение репозитория — обеспечение приложению стабильного API методов CRUD и реализация данных методов.

CRUD (сокр. от англ. **create**, **read**, **update**, **delete** — «создать, прочесть, обновить, удалить») — акроним, обозначающий четыре базовые функции, используемые при работе с персистентными хранилищами данных.

Пример создания и использования репозитория приложении ASP.NET Core 3.0 Razor Pages и ORM Entity Framwork Core

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplication6.Models
{

    public class User
    {

        public Guid Id { get; set; }

        public string Name { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

namespace WebApplication6.Models
{
    public class WebApplication6Context : DbContext
    {
        public WebApplication6Context (DbContextOptions<WebApplication6Context> options)
            : base(options)
        {
        }


        public DbSet<WebApplication6.Models.User> Users { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

namespace WebApplication6.Models
{
    Ссылок: 9
    public class UserRepository
    {
        private readonly WebApplication6Context _context;
        Ссылок: 4
        public UserRepository(WebApplication6Context context)
        {
            _context = context;
        }
```

```
ссылка: 1
public async Task<List<User>> GetAllUsersAsync()
{
    return await _context.Users.ToListAsync();
}

ссылка: 1
public async Task AddUserAsync(User user)
{
    await _context.Users.AddAsync(user);
    await _context.SaveChangesAsync();
}

Ссылок: 2
public async Task<User> GetUserAsync(Guid id)
{
    return await _context.Users.FindAsync(id);
}
```

```
public async Task<User> GetUserAsync(Guid id)
{
    return await _context.Users.FindAsync(id);
}

ссылка: 1
public async Task UpdateUserAsync(User user)
{
    User existUser = await _context.Users.FindAsync(user.Id);
    _context.Entry(existUser).CurrentValues.SetValues(user);
    await _context.SaveChangesAsync();

}
```

```
        ссылка: 1
        public async Task<bool> DeleteUserAsync(Guid id)
        {
            User user = await _context.Users.FindAsync(id);
            if (user == null)
            {
                return false;
            }
            else
            {
                _context.Users.Remove(user);
                await _context.SaveChangesAsync();
                return true;
            }
        }
    }
}
```

Index

Create New

Name	
User 1	Edit | Details | Delete
User 2	Edit | Details | Delete
User 3	Edit | Details | Delete

```
@page
@model WebApplication6.Pages.Users.IndexModel

@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

<p>
    <a asp-page="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Users[0].Name)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
```

```
@foreach (var item in Model.Users) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                <a asp-page="./Edit" asp-route-id="@item.Id">Edit</a> |
                <a asp-page="./Details" asp-route-id="@item.Id">Details</a> |
                <a asp-page="./Delete" asp-route-id="@item.Id">Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>
```

```
namespace WebApplication6.Pages.Users
{
    Ссылок: 6
    public class IndexModel : PageModel
    {
        private readonly UserRepository userRepository;

        Ссылок: 0
        public IndexModel(WebApplication6.Models.WebApplication6Context context)
        {
            userRepository = new UserRepository(context);
        }

        Ссылок: 3
        public IList<User> Users { get;set; }

        Ссылок: 0
        public async Task OnGetAsync()
        {
            Users = await userRepository.GetAllUsersAsync();
        }
    }
}
```

Create

User

Name

Create

Back to List

```razor
@page
@model WebApplication6.Pages.Users.CreateModel

@{
    ViewData["Title"] = "Create";
}

<h1>Create</h1>

<h4>User</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="User.Name" class="control-label"></label>
                <input asp-for="User.Name" class="form-control" />
                <span asp-validation-for="User.Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-page="Index">Back to List</a>
</div>

```

```
Ссылок: 6
public class CreateModel : PageModel
{
    private readonly UserRepository userRepository;

    Ссылок: 0
    public CreateModel(WebApplication6.Models.WebApplication6Context context)
    {
        userRepository = new UserRepository(context);
    }

    Ссылок: 0
    public IActionResult OnGet()
    {
        return Page();
    }

    [BindProperty]
    Ссылок: 4
    public User User { get; set; }
```

```
        ссылок: 0
        public async Task<IActionResult> OnPostAsync()
        {
            if (!ModelState.IsValid)
            {
                return Page();
            }

            await userRepository.AddUserAsync(User);

            return RedirectToPage("./Index");
        }
    }
}
```

Delete

Are you sure you want to delete this?

User

Name User 2

Delete | Back to List

```razor
@page
@model WebApplication6.Pages.Users.DeleteModel

@{
    ViewData["Title"] = "Delete";
}

<h1>Delete</h1>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>User</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.User.Name)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.User.Name)
        </dd>
    </dl>

    <form method="post">
        <input type="hidden" asp-for="User.Id" />
        <input type="submit" value="Delete" class="btn btn-danger" /> |
        <a asp-page="./Index">Back to List</a>
    </form>
</div>

```

```
public class DeleteModel : PageModel
{
    private readonly UserRepository userRepository;

    Ссылок: 0
    public DeleteModel(WebApplication6.Models.WebApplication6Context context)
    {
        userRepository = new UserRepository(context);
    }

    [BindProperty]
    Ссылок: 5
    public User User { get; set; }

    Ссылок: 0
    public async Task<IActionResult> OnGetAsync(Guid? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        User = await userRepository.GetUserAsync((Guid)id);

        if (User == null)
        {
            return NotFound();
        }
        return Page();
    }
```

```
        Ссылок: 0
        public async Task<IActionResult> OnPostAsync(Guid? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            if(await userRepository.DeleteUserAsync((Guid)id))
            {
                return RedirectToPage("./Index");
            }
            else
            {
                return NotFound();
            }

        }
    }
}
```

Edit

User

Name

User 2

Save

Back to List

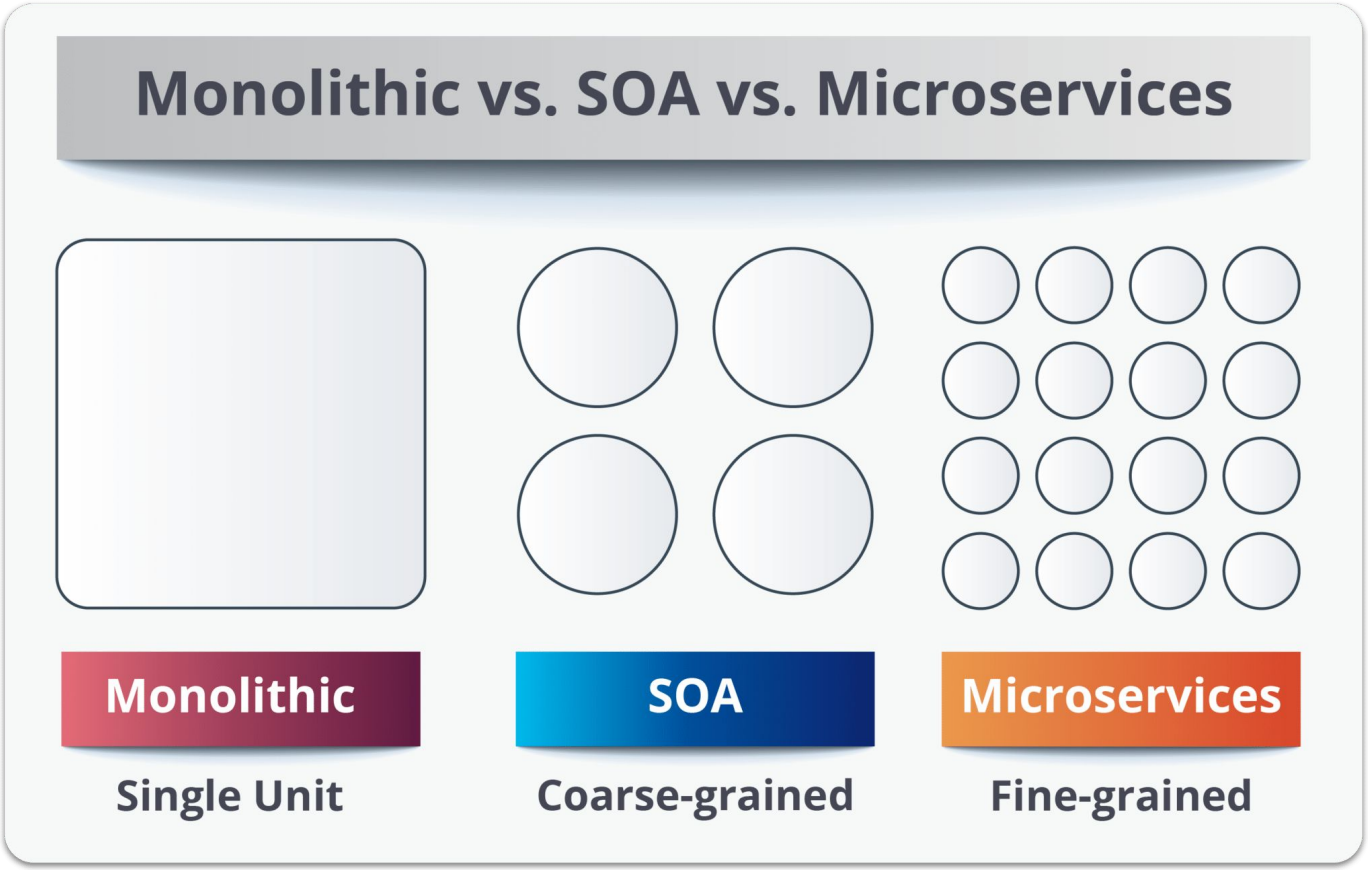
```
@page
@model WebApplication6.Pages.Users.EditModel

@{
    ViewData["Title"] = "Edit";
}

<h1>Edit</h1>

<h4>User</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="User.Id" />
            <div class="form-group">
                <label asp-for="User.Name" class="control-label"></label>
                <input asp-for="User.Name" class="form-control" />
                <span asp-validation-for="User.Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-page="./Index">Back to List</a>
</div>

```

```
public class EditModel : PageModel
{
    private readonly UserRepository userRepository;

    Ссылок: 0
    public EditModel(WebApplication6.Models.WebApplication6Context context)
    {
        userRepository = new UserRepository(context);
    }

    [BindProperty]
    Ссылок: 7
    public User User { get; set; }

    Ссылок: 0
    public async Task<IActionResult> OnGetAsync(Guid? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        User = await userRepository.GetUserAsync((Guid)id);

        if (User == null)
        {
            return NotFound();
        }
        return Page();
    }
```

```
        public async Task<IActionResult> OnPostAsync()
        {
            if (!ModelState.IsValid)
            {
                return Page();
            }
            await userRepository.UpdateUserAsync(User);

            return RedirectToPage("./Index");
        }
    }
}
```

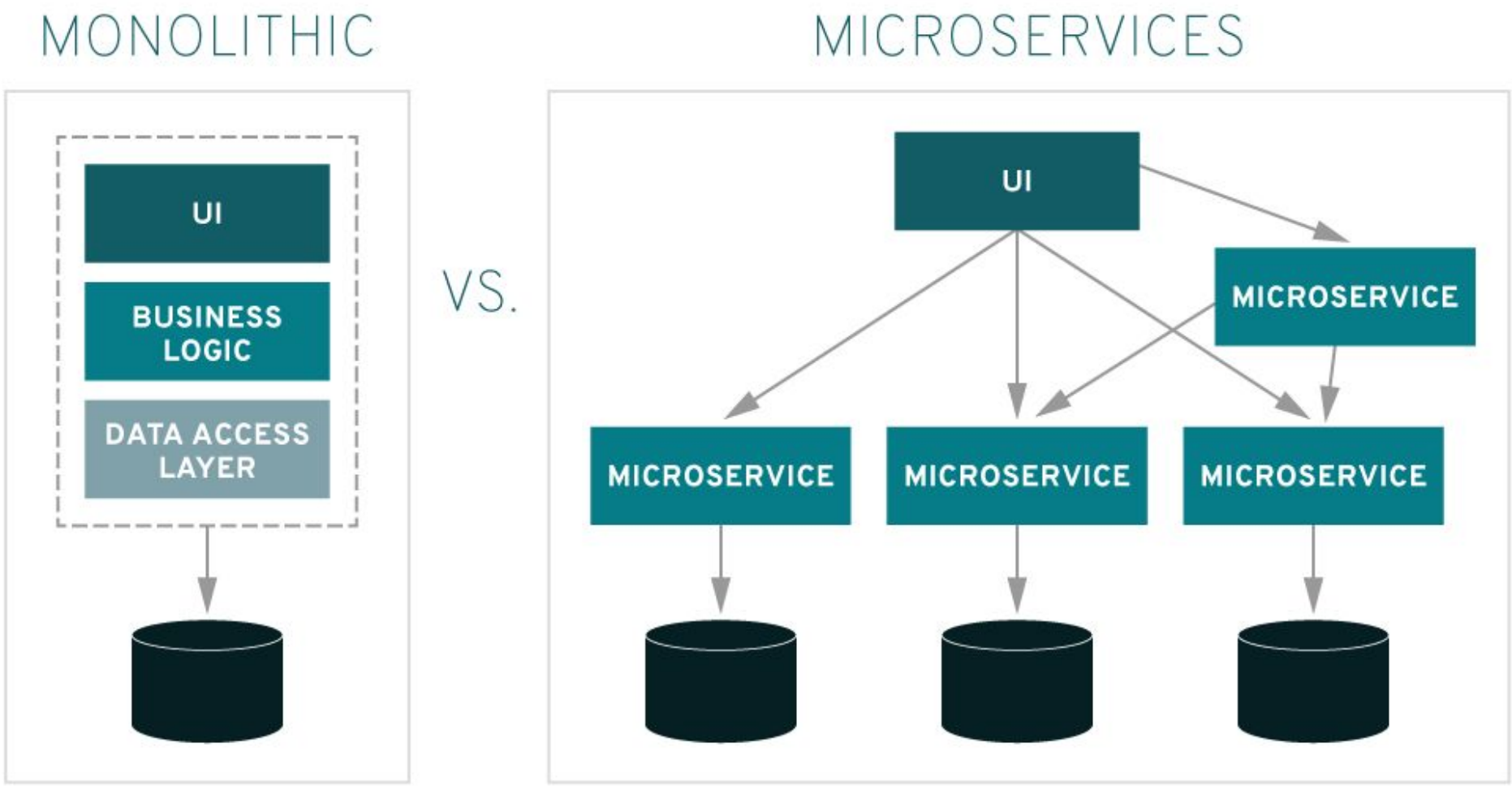
Использование микрослужб с шаблонами DDD

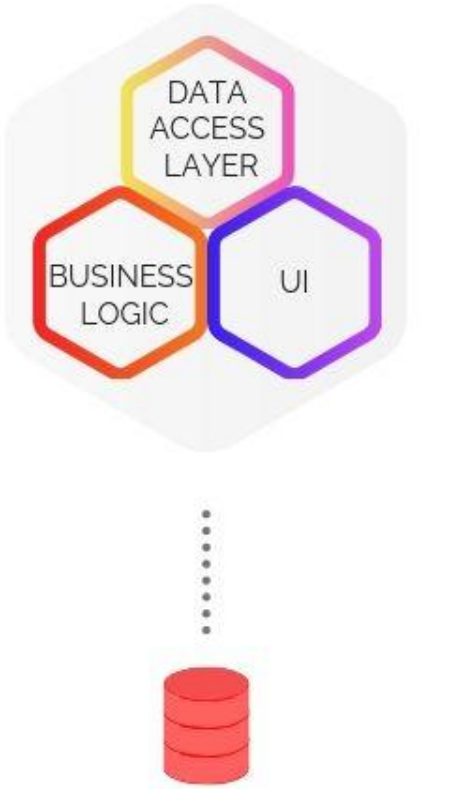
Микросервисная архитектура — вариант сервис-ориентированной архитектуры программного обеспечения, направленный на взаимодействие небольших, слабо связанных и легко изменяемых модулей — микросервисов.



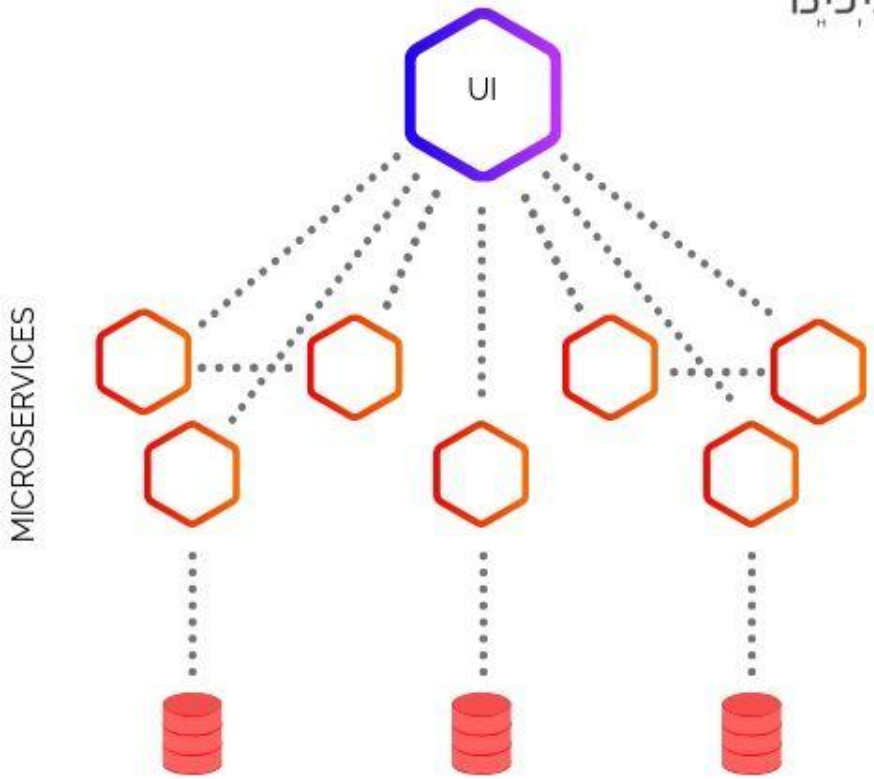
Свойства, характерные для микросервисной архитектуры:

- модули можно легко заменить в любое время: акцент на простоту, независимость развёртывания и обновления каждого из микросервисов;
- модули организованы вокруг функций: микросервис по возможности выполняет только одну достаточно элементарную функцию;
- модули могут быть реализованы с использованием различных языков программирования, фреймворков, связующего программного обеспечения, выполняться в различных средах контейнеризации, виртуализации, под управлением различных операционных систем на различных аппаратных платформах: приоритет отдаётся в пользу наибольшей эффективности для каждой конкретной функции, нежели стандартизации средств разработки и исполнения;
- архитектура симметричная, а не иерархическая: зависимости между микросервисами одноранговые.


MONOLITHIC
UI
BUSINESS LOGIC
DATA ACCESS LAYER
VS.
MICROSERVICES
UI
MICROSERVICE
MICROSERVICE
MICROSERVICE
MICROSERVICE



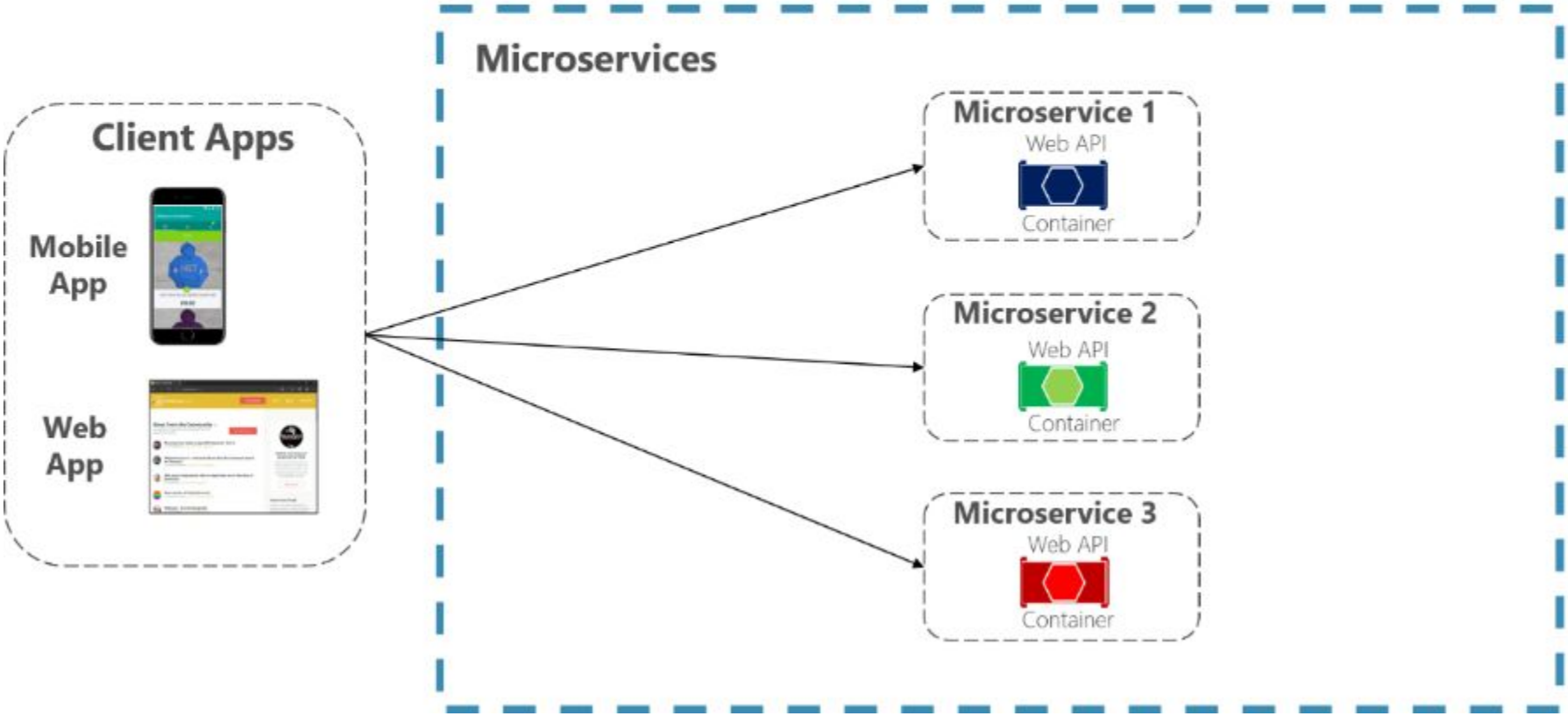
Monolithic Architecture



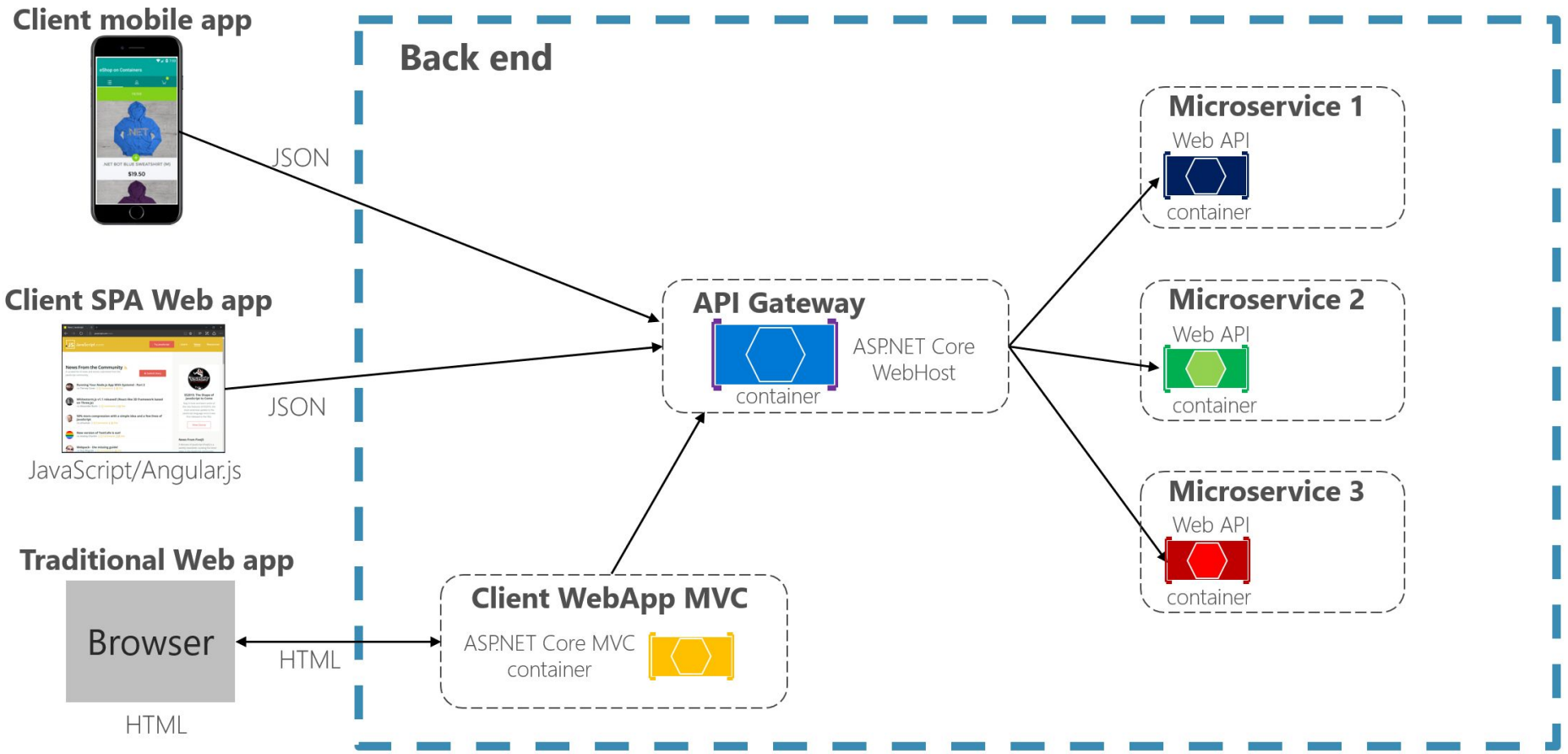
Microservices Architecture

Direct Client-To-Microservice communication

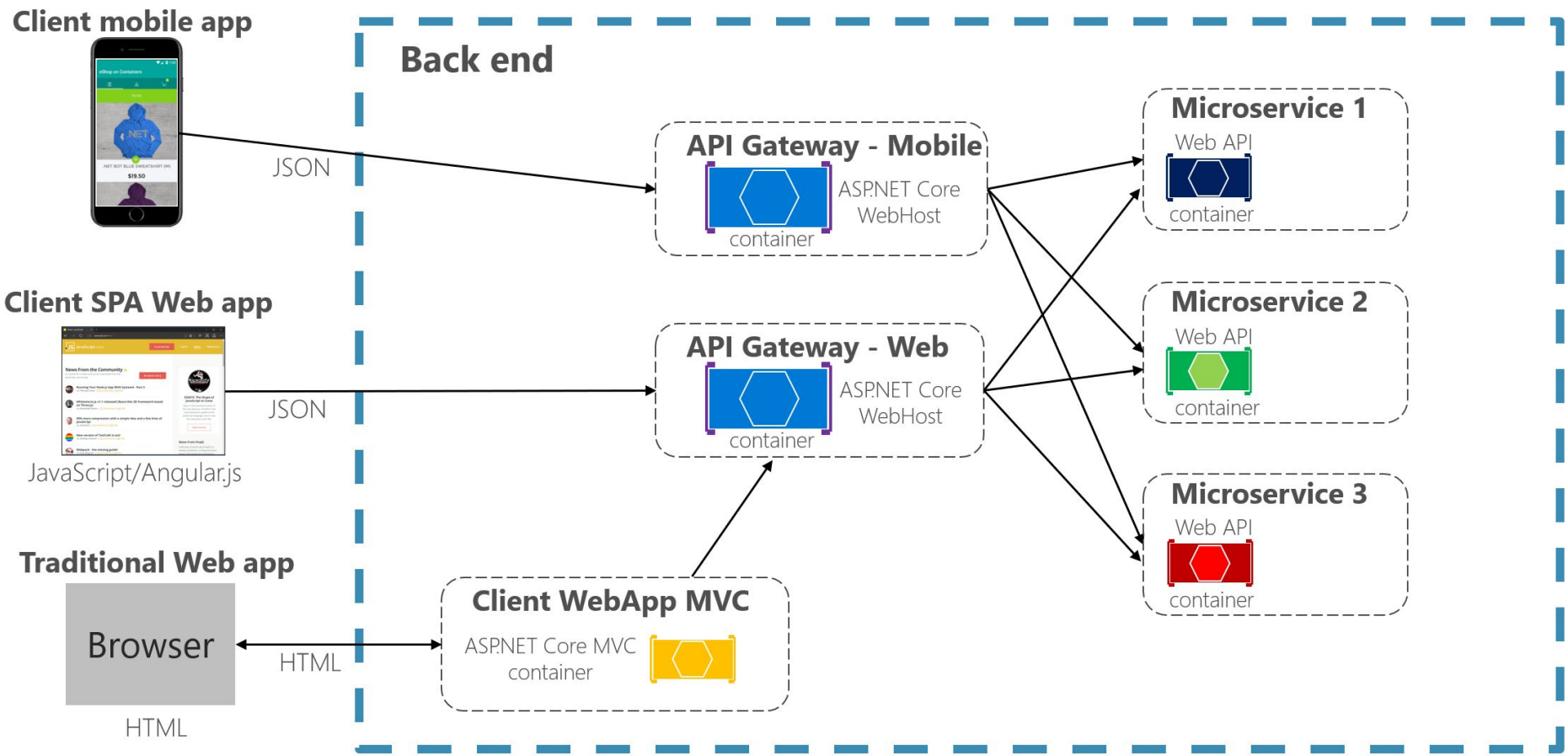
Architecture



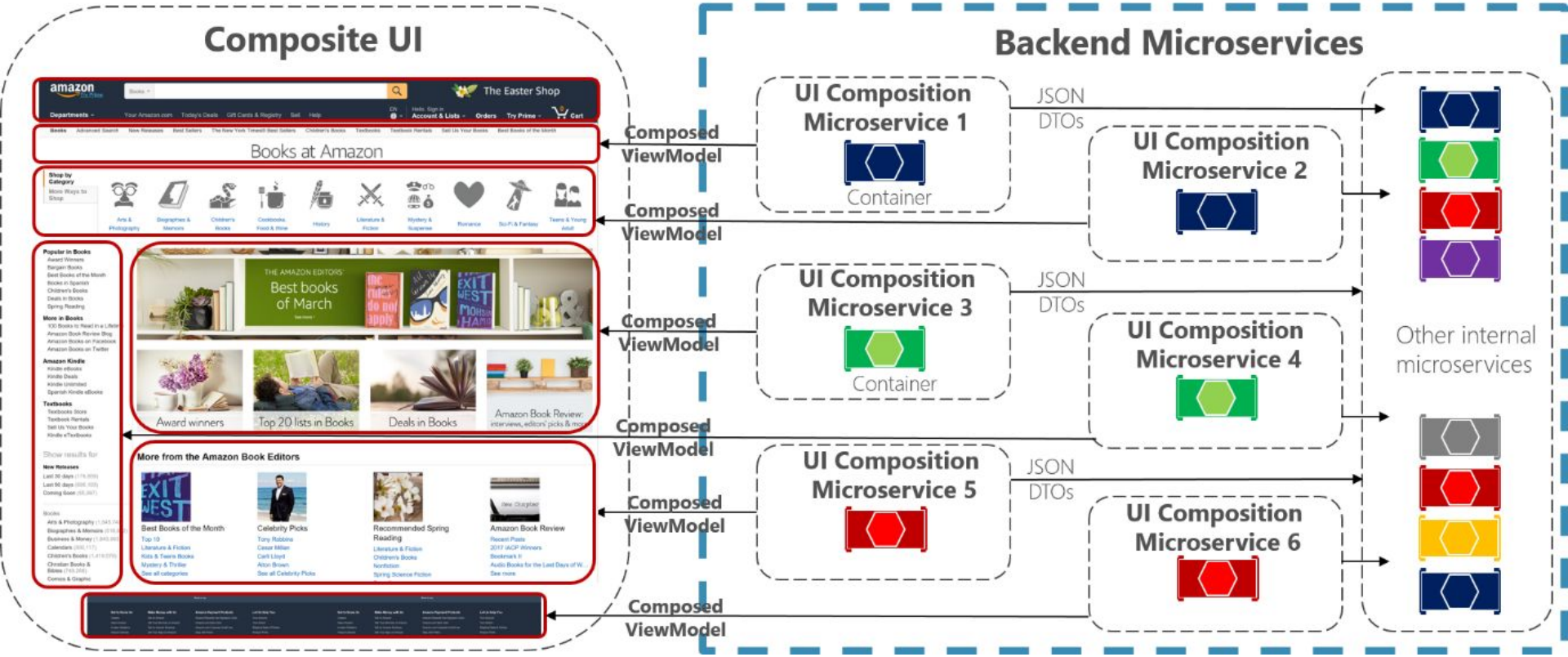
Using a single custom **API Gateway service**

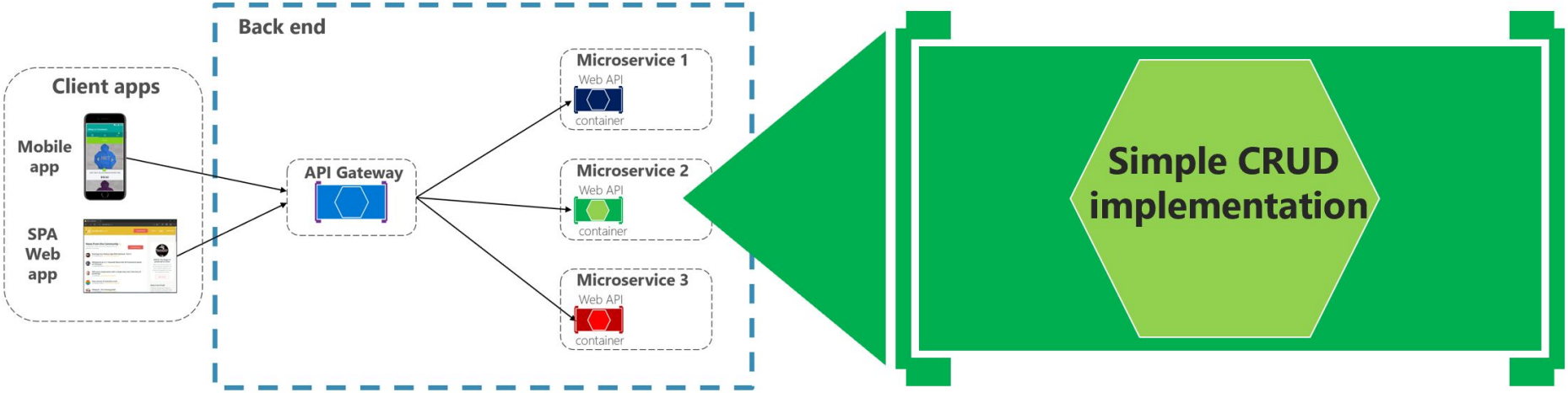


Using multiple **API Gateways / BFF**



Composite UI *generated* by microservices

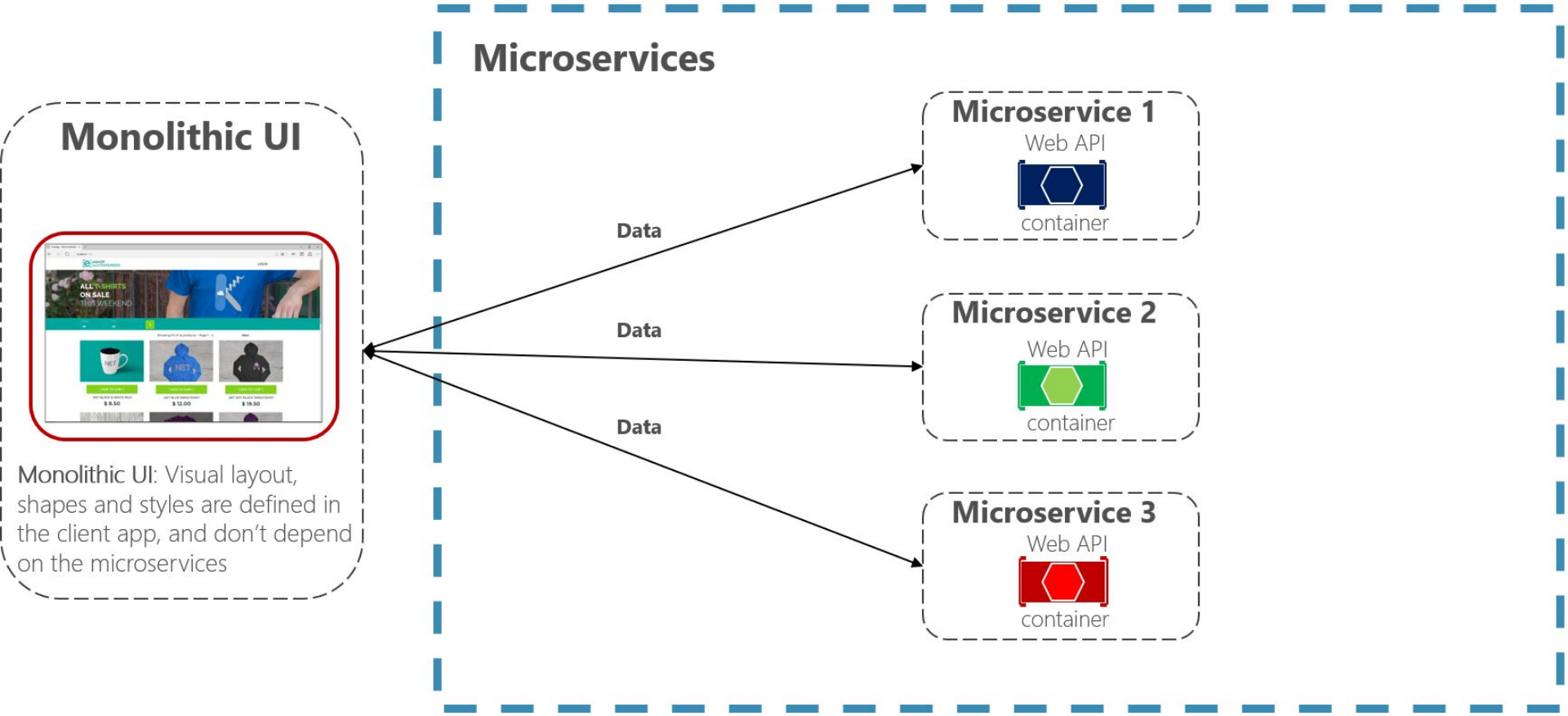




- External microservice patterns
- API Gateway
- Resilient communication
- Pub/Sub and event driven

Internal design patterns per microservice

Monolithic UI consuming microservices



CQRS (Command Query Responsibility Segregation)

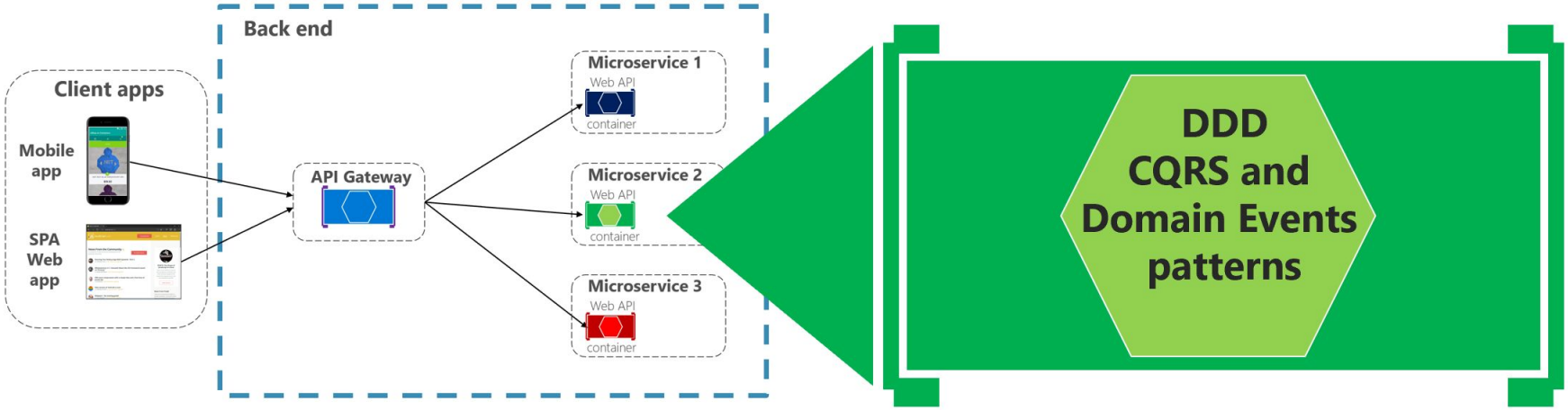
Любой метод должен или изменять состояние объекта, или возвращать результат — но не то и другое одновременно.

CQRS — архитектурный паттерн, который предусматривает разделение приложения на две отдельные модели данных. Одна из них отвечает за обновление данных (операции по записи или **команды**), другая — за отображение данных (операции по считыванию или запросы).

Команды VS запросы: строгое распределение ролей
Каждая операция — это либо команда или запрос, и не может быть сочетанием обоих.
Команда (также известная как модификатор) вносит изменения в систему, но не возвращает данные. Запрос, наоборот, получает и отображает данные из системы, но не может их менять.

External architecture per application

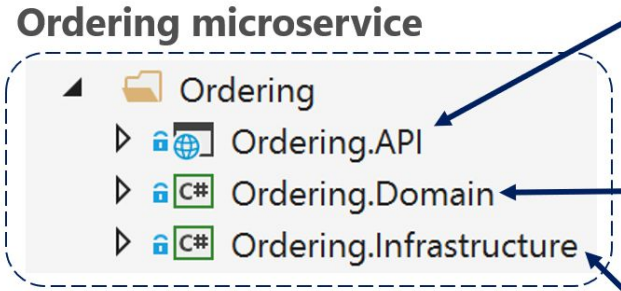
Internal architecture per microservice



- External microservice patterns
- API Gateway
- Resilient communication
- Pub/Sub and event driven

Internal DDD patterns in addition to SOLID principles and Dependency Injection

Layers in a Domain-Driven Design Microservice



Application layer

- ASP.NET Web API
- Network access to microservice
- API contracts/implementation
- Commands and command handlers
- Queries (when using an CQS approach)
  - Micro ORMs like Dapper

Domain model layer

- Domain entity model
- POCO entity classes (clean C# code)
- Domain entities with data + behavior
- DDD patterns:
  - Domain entity, aggregate
  - Aggregate root, value object
  - Repository contracts/interfaces

Infrastructure layer

- Data persistence infrastructure
  - Repository implementation
- Use of ORMs or data access API:
  - Entity Framework Core or any ORM
  - ADO.NET
  - Any NoSQL database API
- Other infrastructure implementation used from the application layer
  - Logging, cryptography, search engine, etc.