

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**

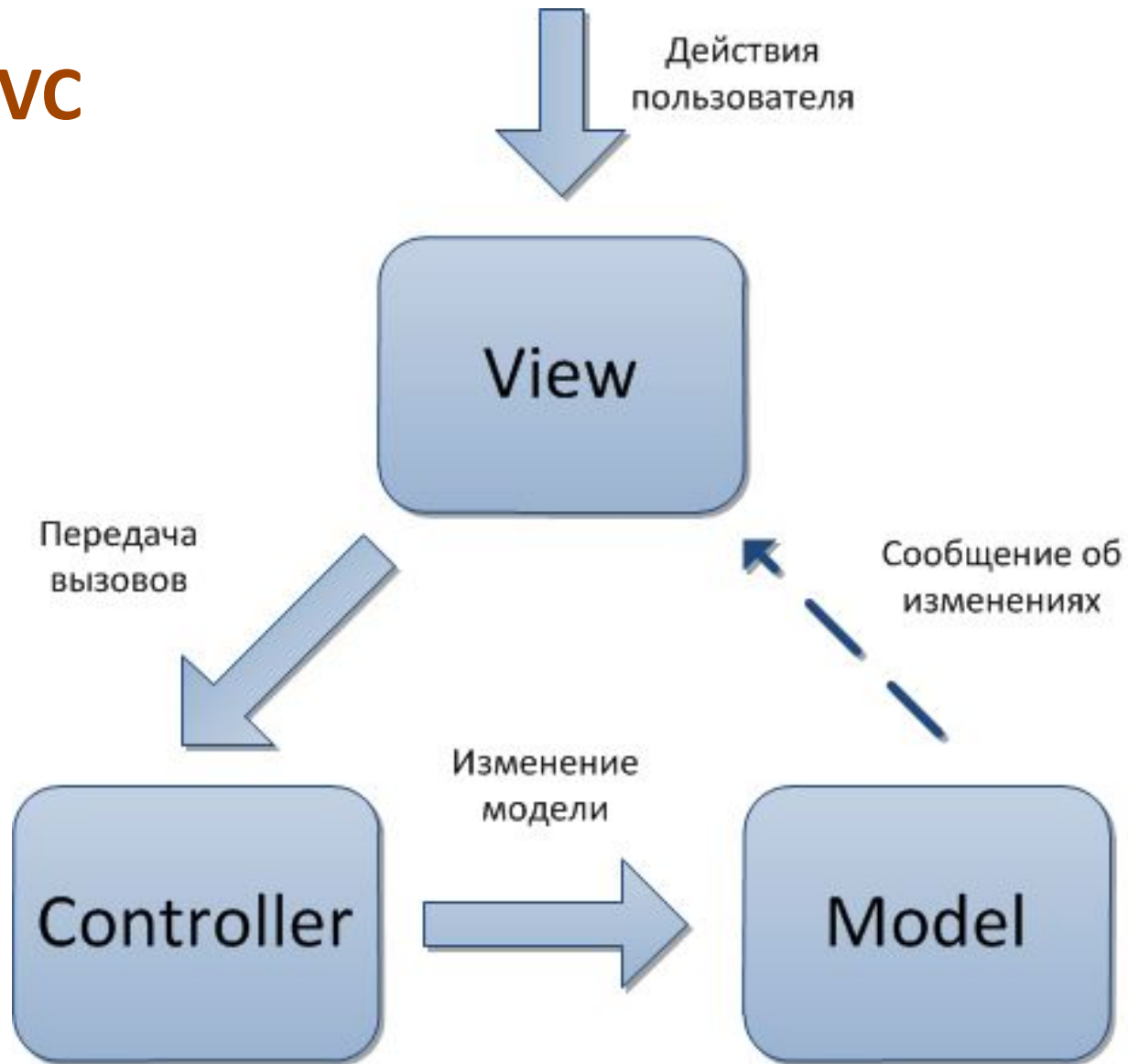
2019

РАЗДЕЛ 2

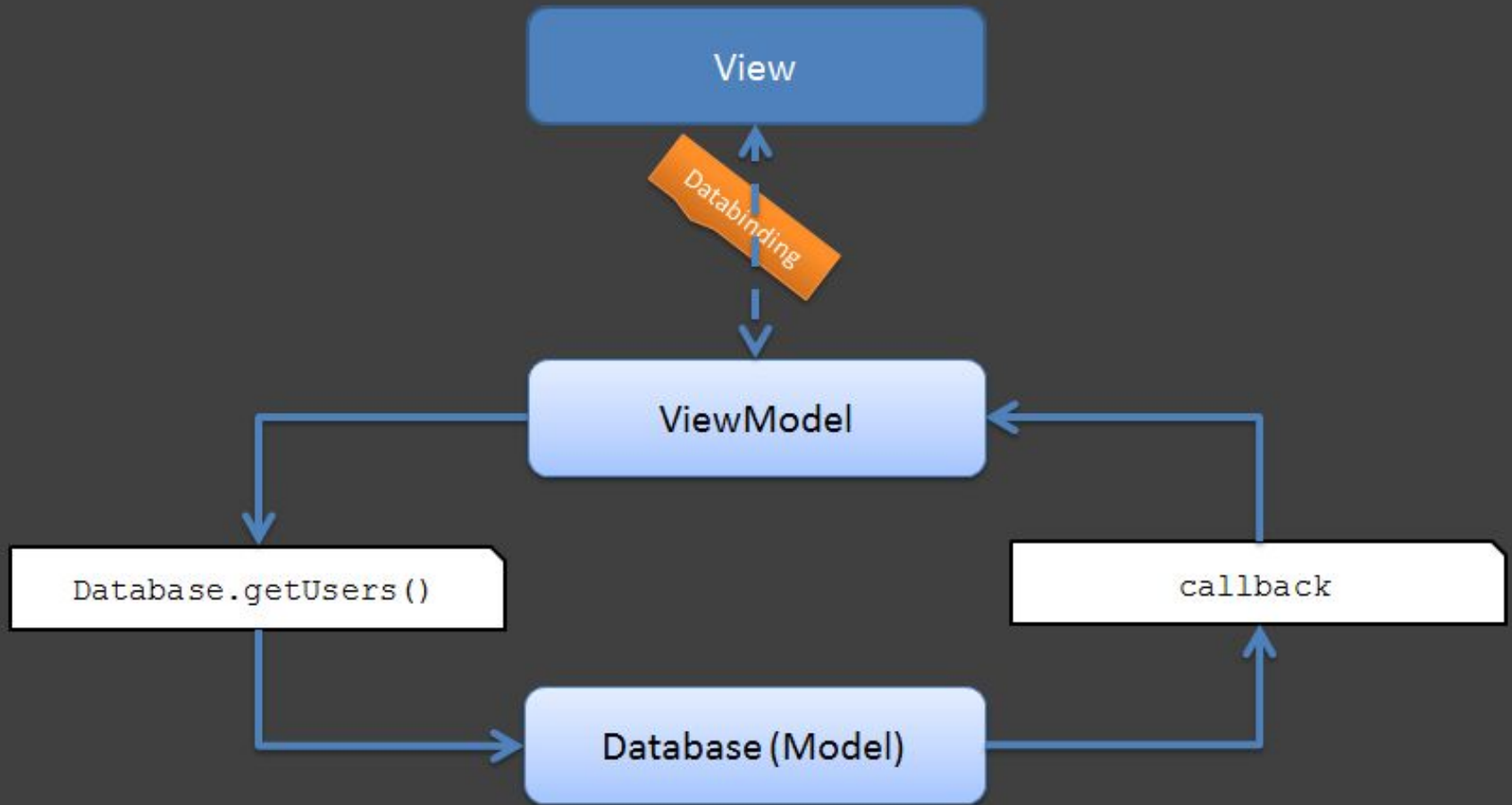
ТЕХНОЛОГИИ DDD

Архитектурные решения DDD

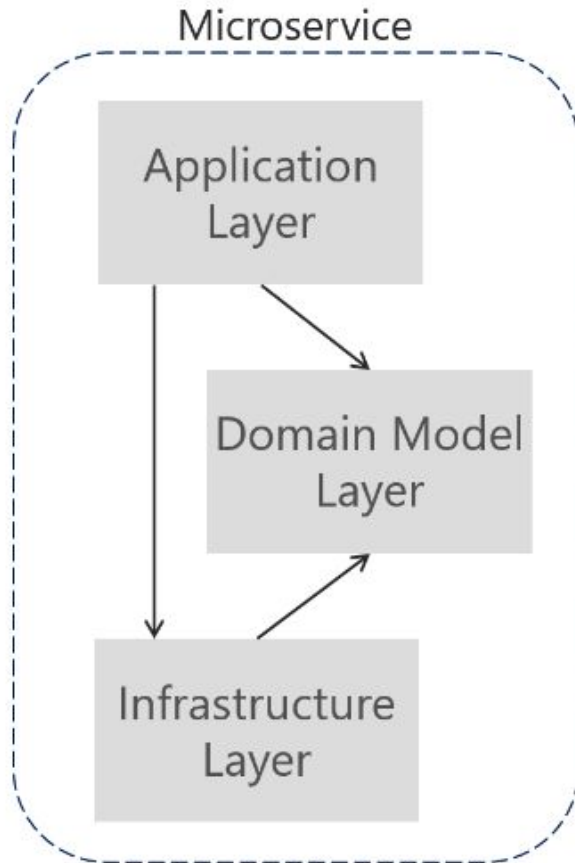
MVC



MVVM



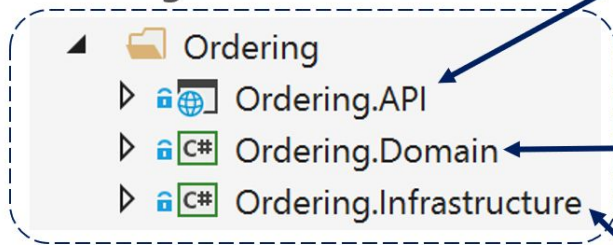
Dependencies between Layers in a Domain-Driven Design service



- 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

Ordering microservice



Application layer

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

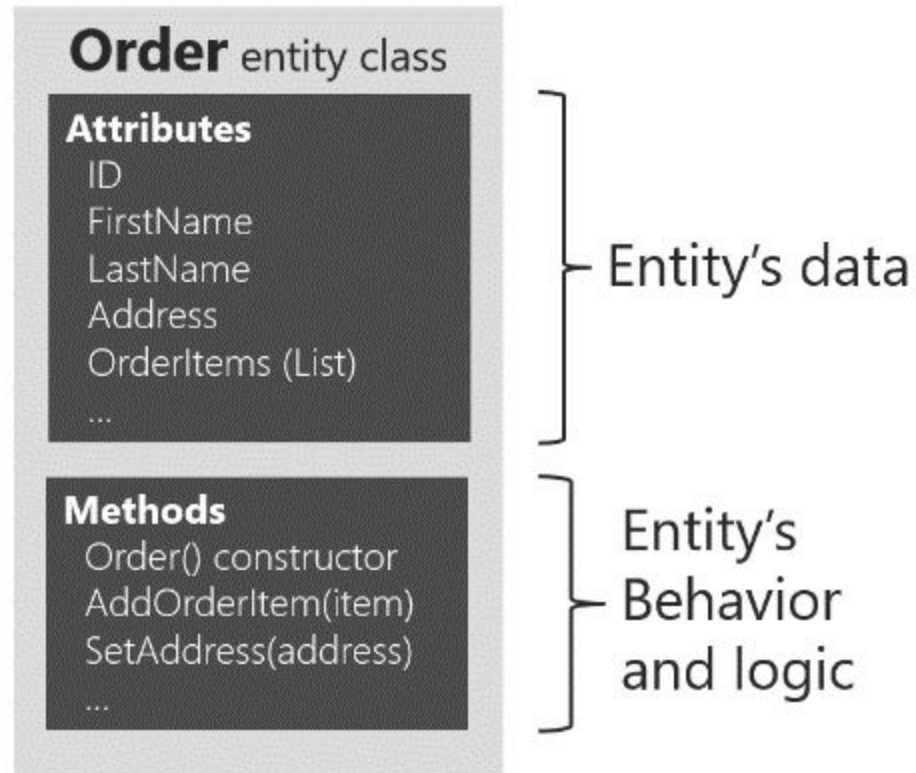
Domain model layer

- 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

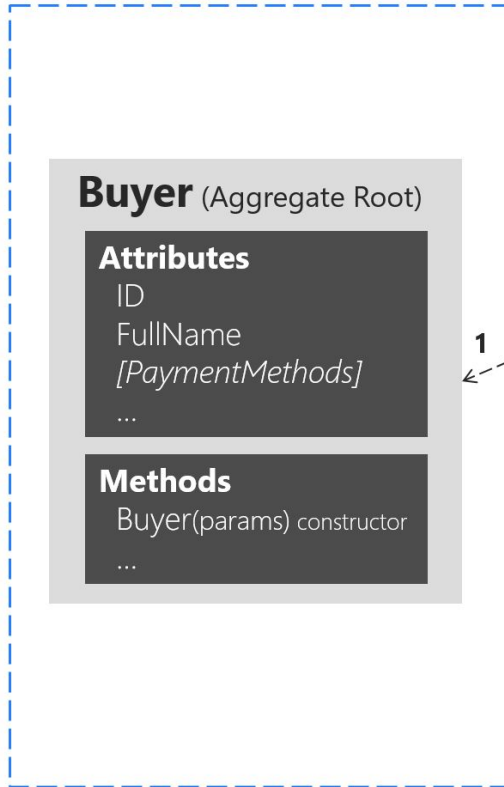
- Data persistence infrastructure
 - Repository implementation
- Use of ORMs or data access API:
 - Entity Framework Core or any ORM
 - ADO.NET
 - Any NoSQL database API
- Other infrastructure implementation used from the application layer
 - Logging, cryptography, search engine, etc.

Domain Entity pattern

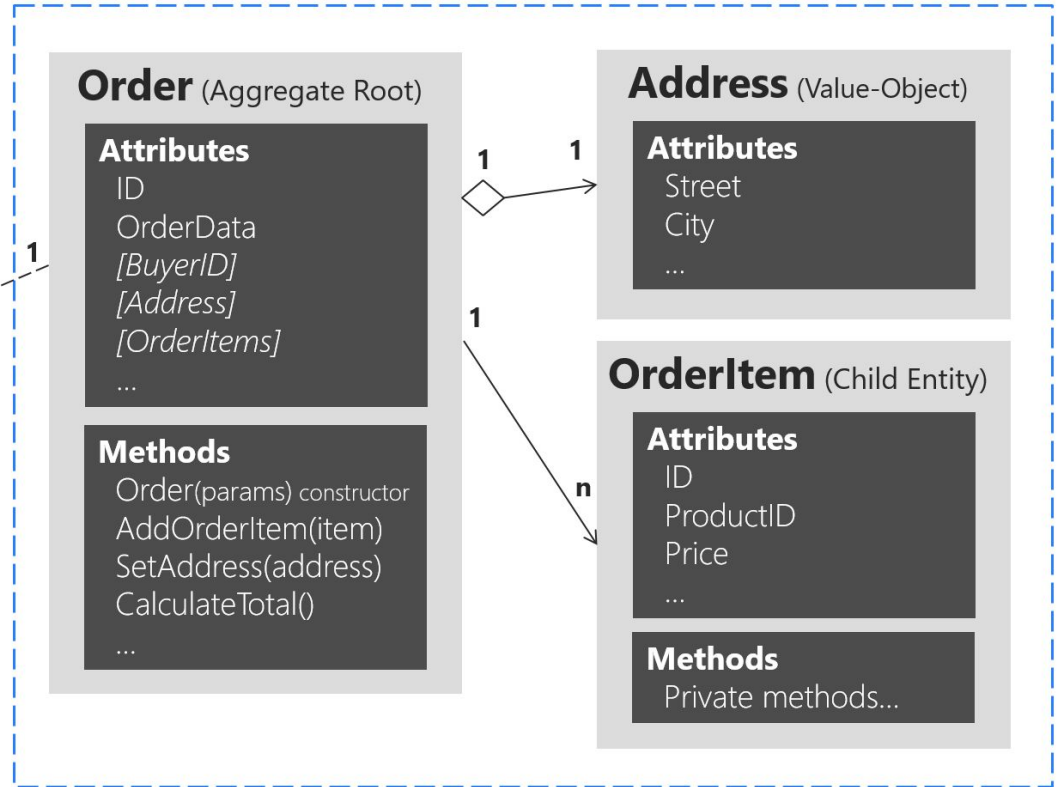


Aggregate pattern

Buyer Aggregate (One entity)



Order Aggregate (Multiple entities and Value-Object)



Объектно-реляционное преобразование

Класс

Персоналия
ID
Фамилия
Имя
Отчество
ToString()

Объекты

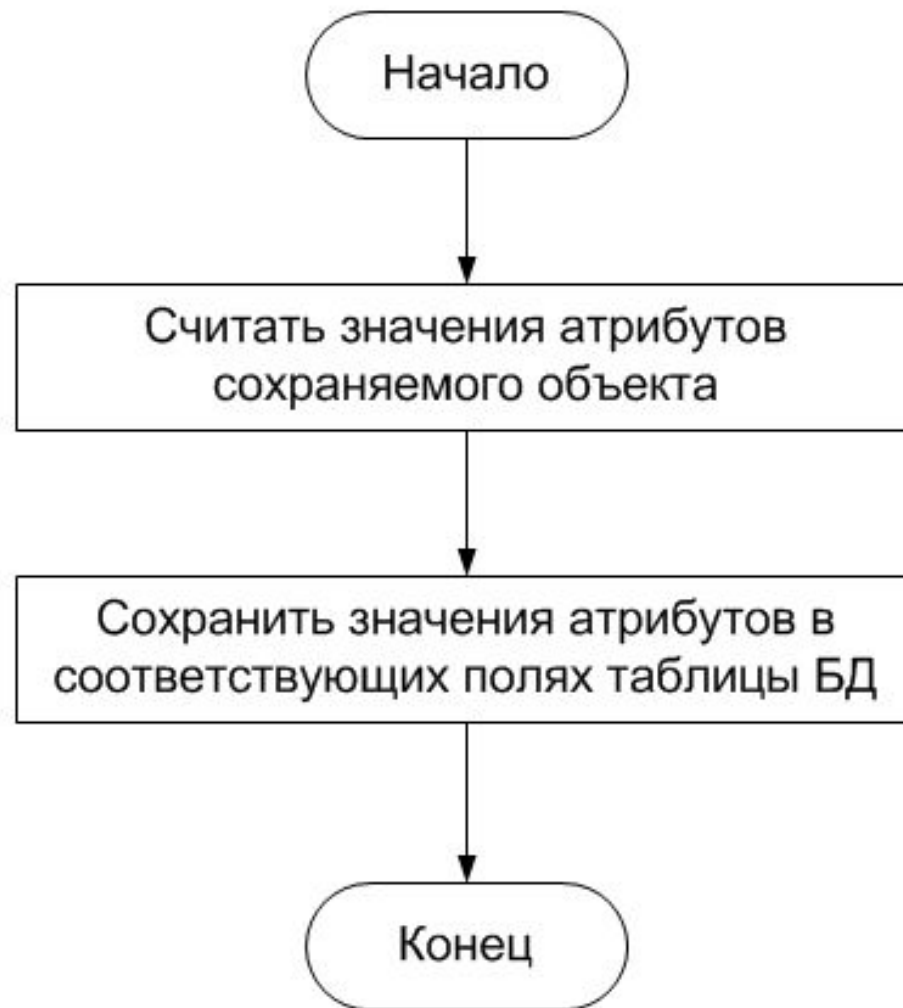
ID = 1
Фамилия = Иванов
Имя = Иван
Отчество = Иванович
ToString()

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



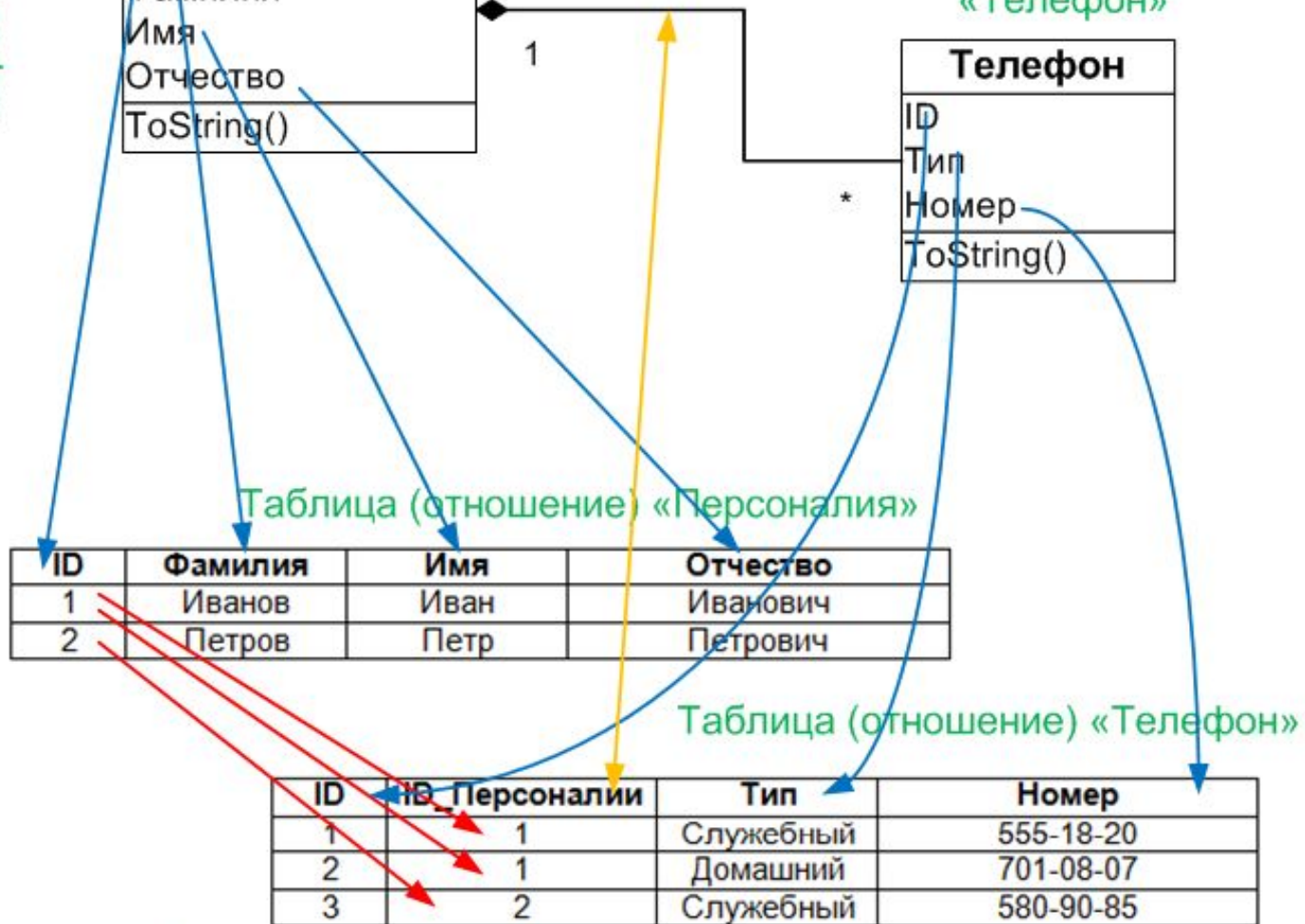
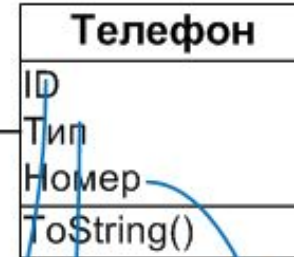


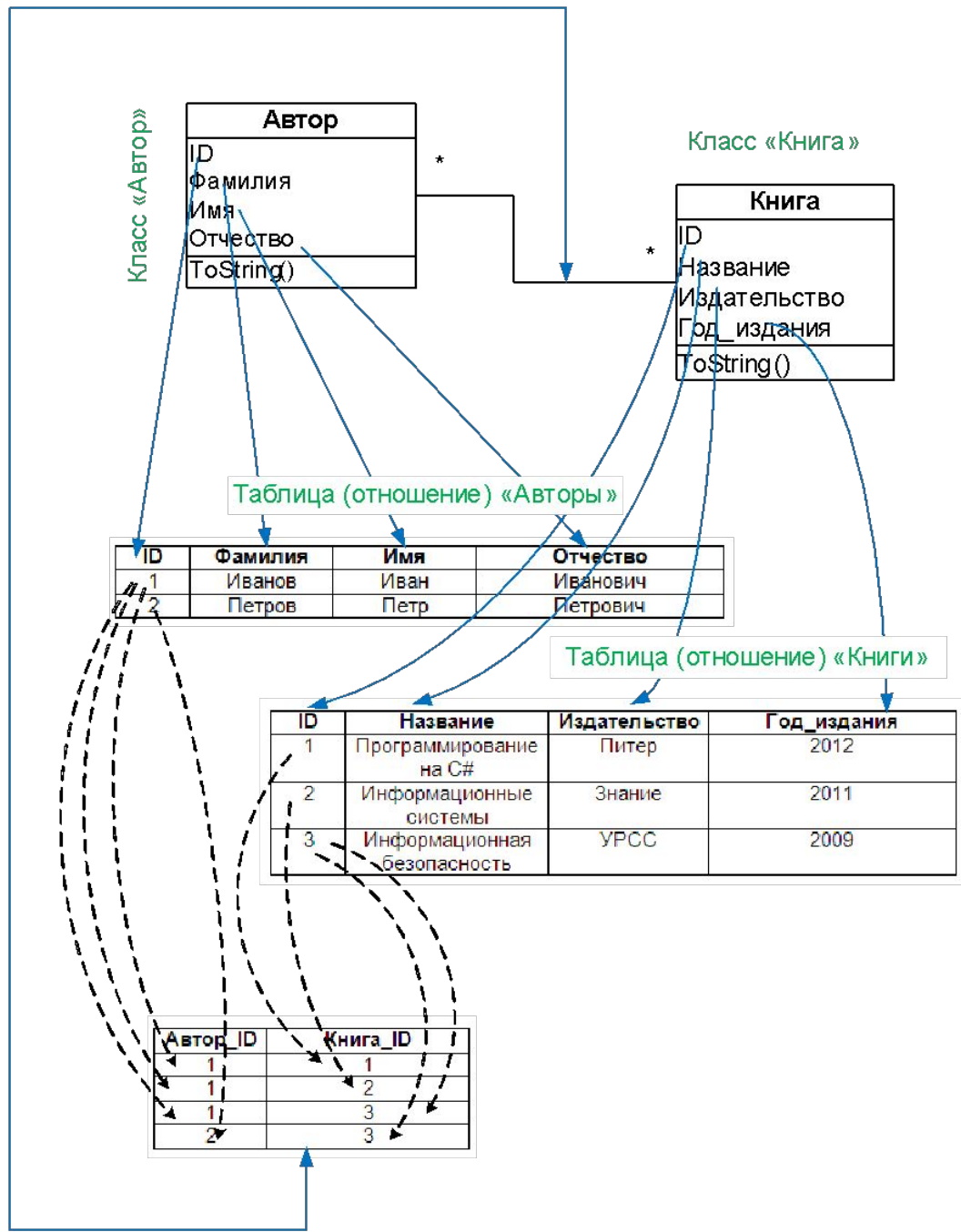


Класс «Персоналия»

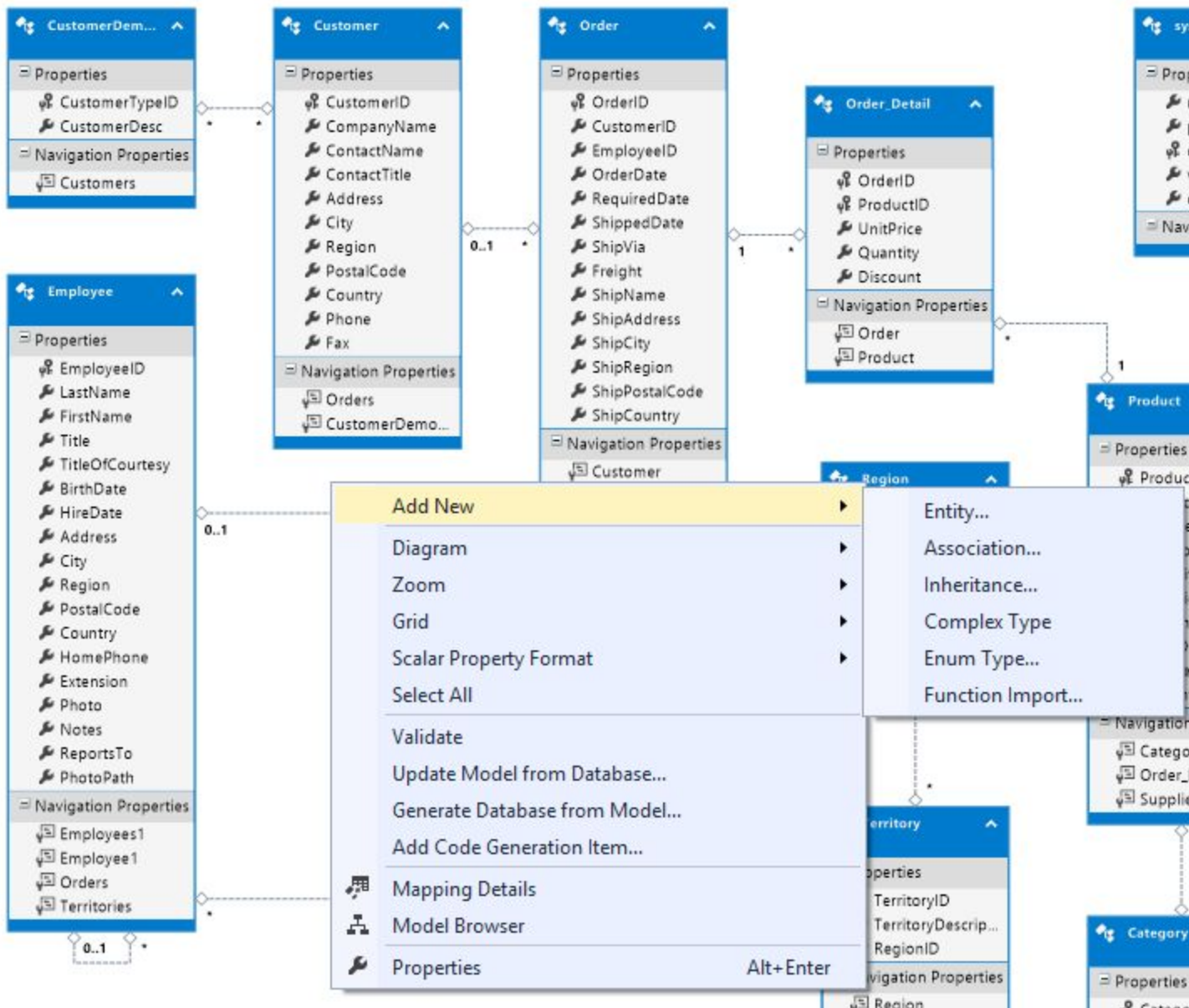


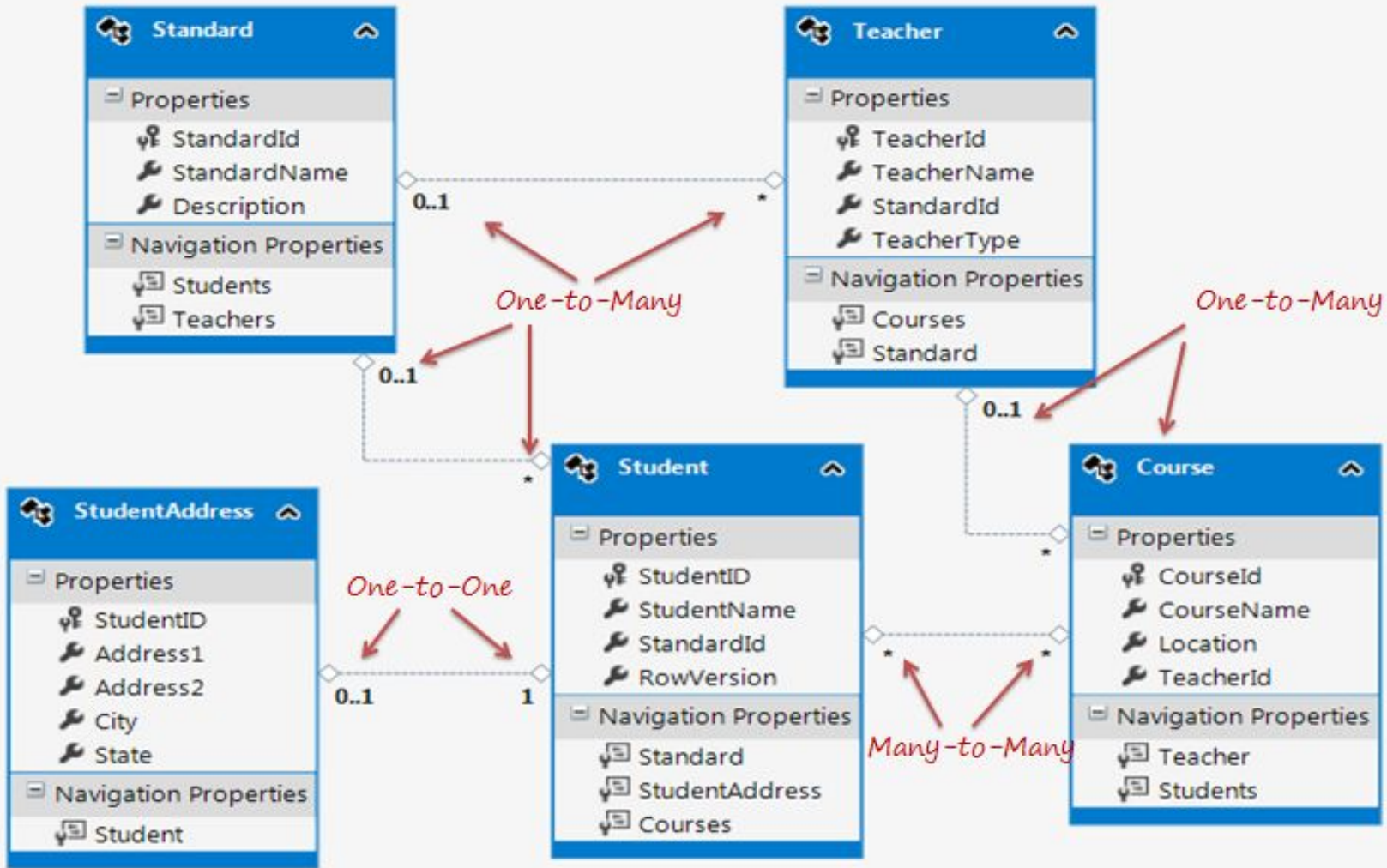
Класс «Телефон»





Entity Framework





**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 Framework
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
```

```
{
```

Ссылка: 9

```
public class UserRepository
```

```
{
```

```
    private readonly WebApplication6Context _context;
```

Ссылка: 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();  
}
```

Ссылка: 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

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

```
1  @page
2  @model WebApplication6.Pages.Users.IndexModel
3
4  @{
5      ViewData["Title"] = "Index";
6  }
7
8  <h1>Index</h1>
9
10 <p>
11     <a asp-page="Create">Create New</a>
12 </p>
13 <table class="table">
14     <thead>
15         <tr>
16             <th>
17                 @Html.DisplayNameFor(model => model.Users[0].Name)
18             </th>
19             <th></th>
20         </tr>
21     </thead>
22     <tbody>
```

```
23  @foreach (var item in Model.Users) {
24      <tr>
25          <td>
26              @Html.DisplayFor(modelItem => item.Name)
27          </td>
28          <td>
29              <a asp-page="./Edit" asp-route-id="@item.Id">Edit</a> |
30              <a asp-page="./Details" asp-route-id="@item.Id">Details</a> |
31              <a asp-page="./Delete" asp-route-id="@item.Id">Delete</a>
32          </td>
33      </tr>
34  }
35  </tbody>
36  </table>
37
```

```
namespace WebApplication6.Pages.Users
```

```
{
```

Ссылка: 6

```
public class IndexModel : PageModel
```

```
{
```

```
    private readonly UserRepository userRepository;
```

Ссылка: 0

```
public IndexModel(WebApplication6.Models.WebApplication6Context context)
```

```
{
```

```
    userRepository = new UserRepository(context);
```

```
}
```

Ссылка: 3

```
public IList<User> Users { get; set; }
```

Ссылка: 0

```
public async Task OnGetAsync()
```

```
{
```

```
    Users = await userRepository.GetAllUsersAsync();
```

```
}
```

```
}
```

```
}
```

Create

User

Name

Create

[Back to List](#)

```
1 @page
2 @model WebApplication6.Pages.Users.CreateModel
3
4 @{
5     ViewData["Title"] = "Create";
6 }
7
8 <h1>Create</h1>
9
10 <h4>User</h4>
11 <hr />
12 <div class="row">
13     <div class="col-md-4">
14         <form method="post">
15             <div asp-validation-summary="ModelOnly" class="text-danger"></div>
16             <div class="form-group">
17                 <label asp-for="User.Name" class="control-label"></label>
18                 <input asp-for="User.Name" class="form-control" />
19                 <span asp-validation-for="User.Name" class="text-danger"></span>
20             </div>
21             <div class="form-group">
22                 <input type="submit" value="Create" class="btn btn-primary" />
23             </div>
24         </form>
25     </div>
26 </div>
27
28 <div>
29     <a asp-page="Index">Back to List</a>
30 </div>
31
```


Ссылка: 6

```
public class CreateModel : PageModel
```

```
{
```

```
    private readonly UserRepository userRepository;
```

Ссылка: 0

```
    public CreateModel(WebApplication6.Models.WebApplication6Context context)
```

```
    {
```

```
        userRepository = new UserRepository(context);
```

```
    }
```

Ссылка: 0

```
    public IActionResult OnGet()
```

```
    {
```

```
        return Page();
```

```
    }
```

```
[BindProperty]
```

Ссылка: 4

```
    public User User { get; set; }
```



CC6/10K. U

```
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](#)

```
1 @page
2 @model WebApplication6.Pages.Users.DeleteModel
3
4 @{
5     ViewData["Title"] = "Delete";
6 }
7
8 <h1>Delete</h1>
9
10 <h3>Are you sure you want to delete this?</h3>
11 <div>
12     <h4>User</h4>
13     <hr />
14     <dl class="row">
15         <dt class="col-sm-2">
16             @Html.DisplayNameFor(model => model.User.Name)
17         </dt>
18         <dd class="col-sm-10">
19             @Html.DisplayFor(model => model.User.Name)
20         </dd>
21     </dl>
22
23     <form method="post">
24         <input type="hidden" asp-for="User.Id" />
25         <input type="submit" value="Delete" class="btn btn-danger" /> |
26         <a asp-page="./Index">Back to List</a>
27     </form>
28 </div>
29
```

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

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

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

    Ссылка: 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();
    }
}
```



Ссылка: 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

Save

[Back to List](#)

```

1  @page
2  @model WebApplication6.Pages.Users.EditModel
3
4  @{
5      ViewData["Title"] = "Edit";
6  }
7
8  <h1>Edit</h1>
9
10 <h4>User</h4>
11 <hr />
12 <div class="row">
13     <div class="col-md-4">
14         <form method="post">
15             <div asp-validation-summary="ModelOnly" class="text-danger"></div>
16             <input type="hidden" asp-for="User.Id" />
17             <div class="form-group">
18                 <label asp-for="User.Name" class="control-label"></label>
19                 <input asp-for="User.Name" class="form-control" />
20                 <span asp-validation-for="User.Name" class="text-danger"></span>
21             </div>
22             <div class="form-group">
23                 <input type="submit" value="Save" class="btn btn-primary" />
24             </div>
25         </form>
26     </div>
27 </div>
28
29 <div>
30     <a asp-page="./Index">Back to List</a>
31 </div>
32

```



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

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

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

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

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

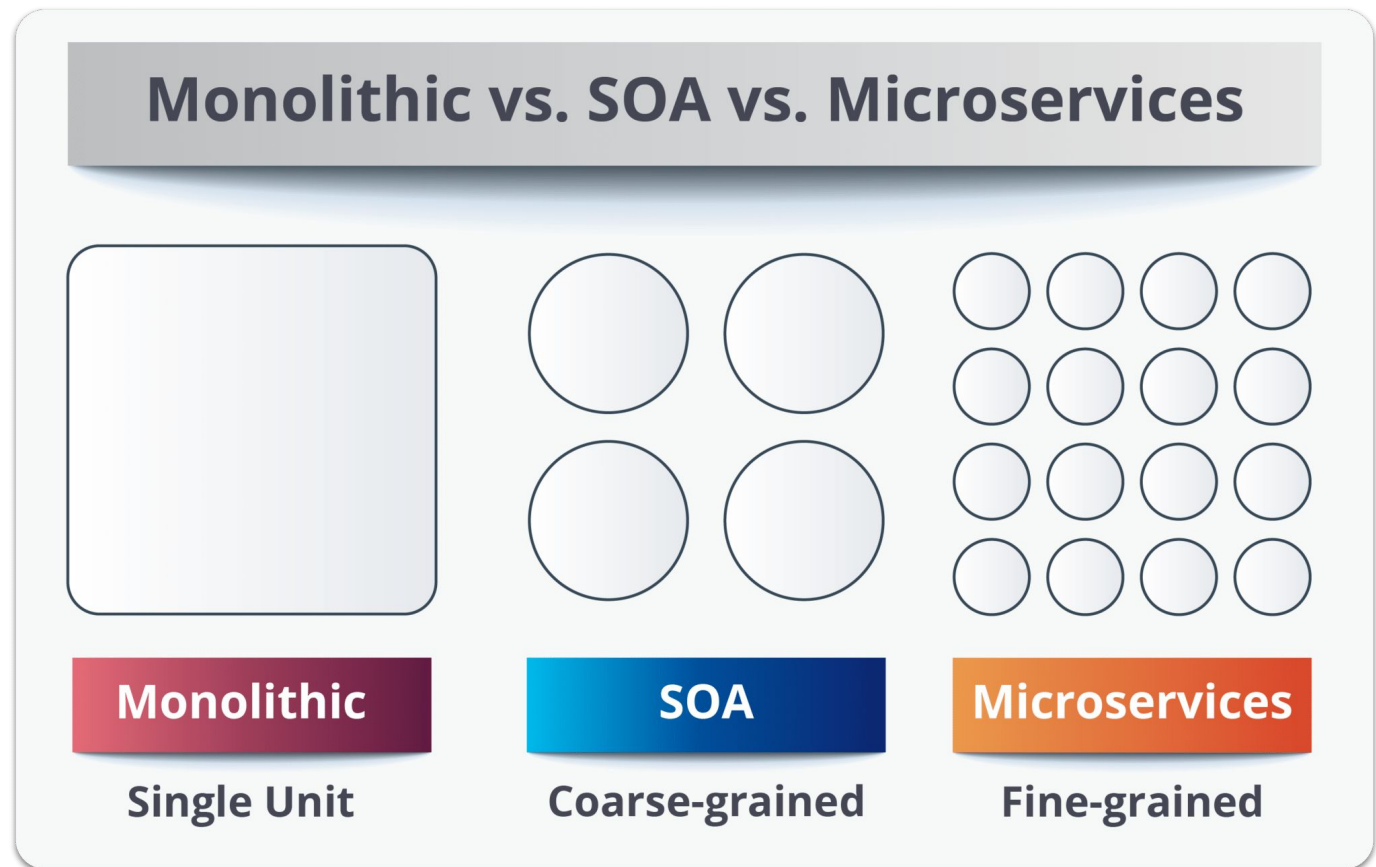
        if (User == null)
        {
            return NotFound();
        }
        return Page();
    }
}
```

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

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

Использование микрослужб с шаблонами DDD

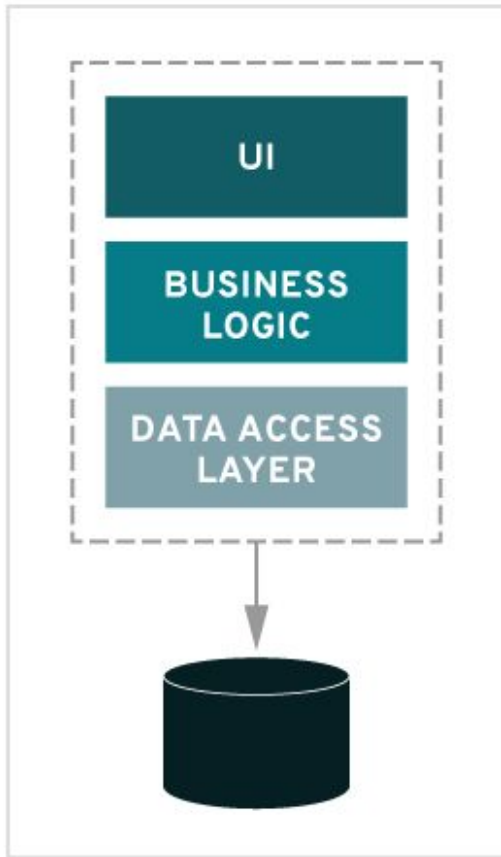
Микросервисная архитектура — вариант сервис-ориентированной архитектуры программного обеспечения, направленный на взаимодействие небольших, слабо связанных и легко изменяемых модулей — микросервисов.



Свойства, характерные для микросервисной архитектуры:

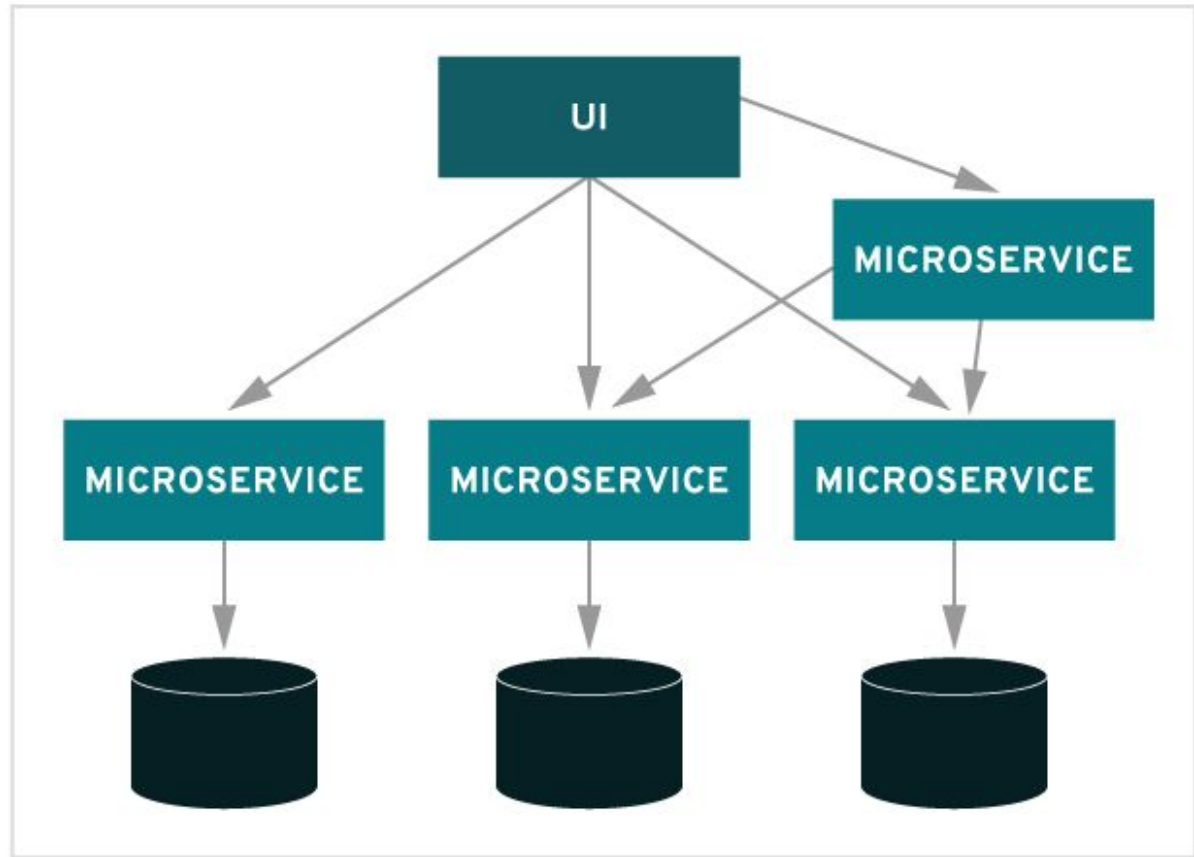
- модули можно легко заменить в любое время: акцент на простоту, независимость развёртывания и обновления каждого из микросервисов;
- модули организованы вокруг функций: микросервис по возможности выполняет только одну достаточно элементарную функцию;
- модули могут быть реализованы с использованием 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



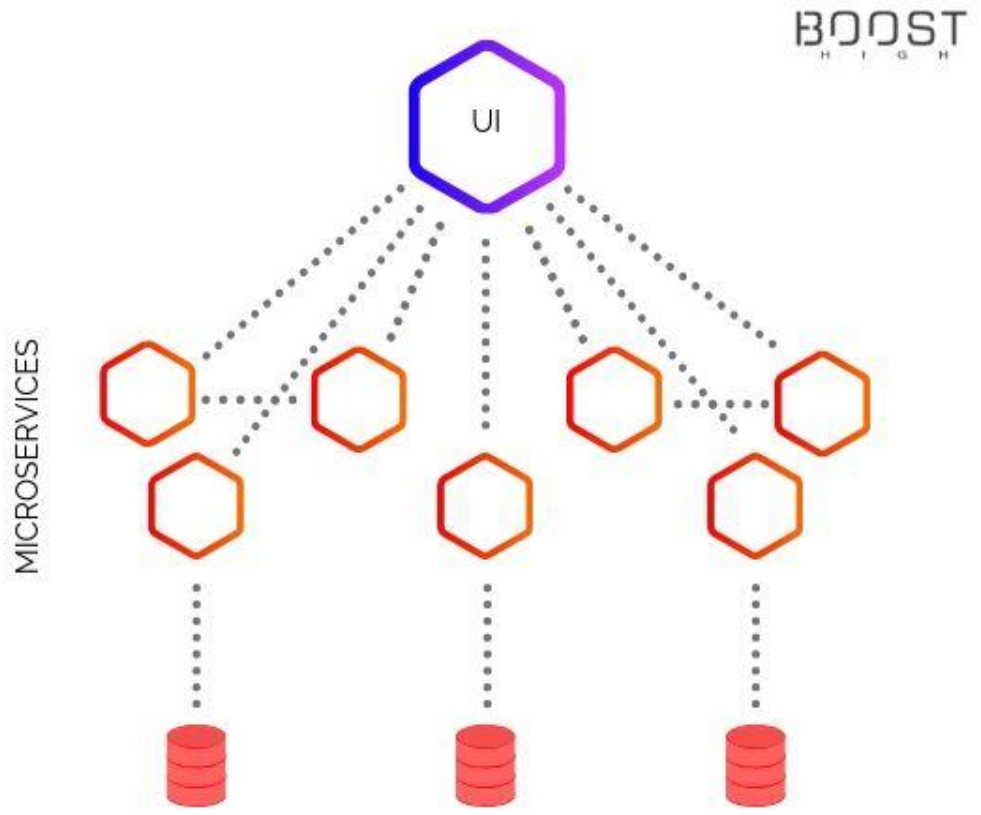
VS.

MICROSERVICES



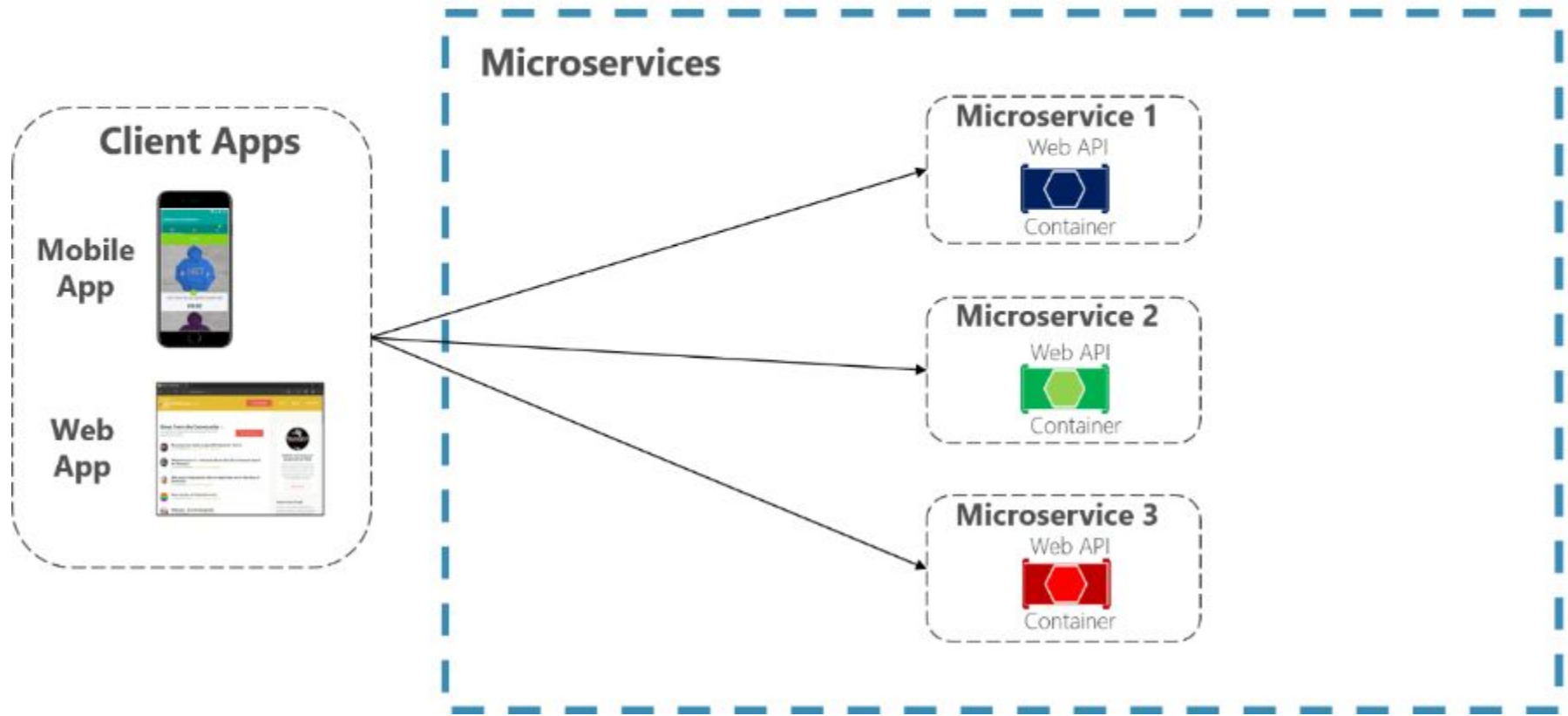


Monolithic Architecture

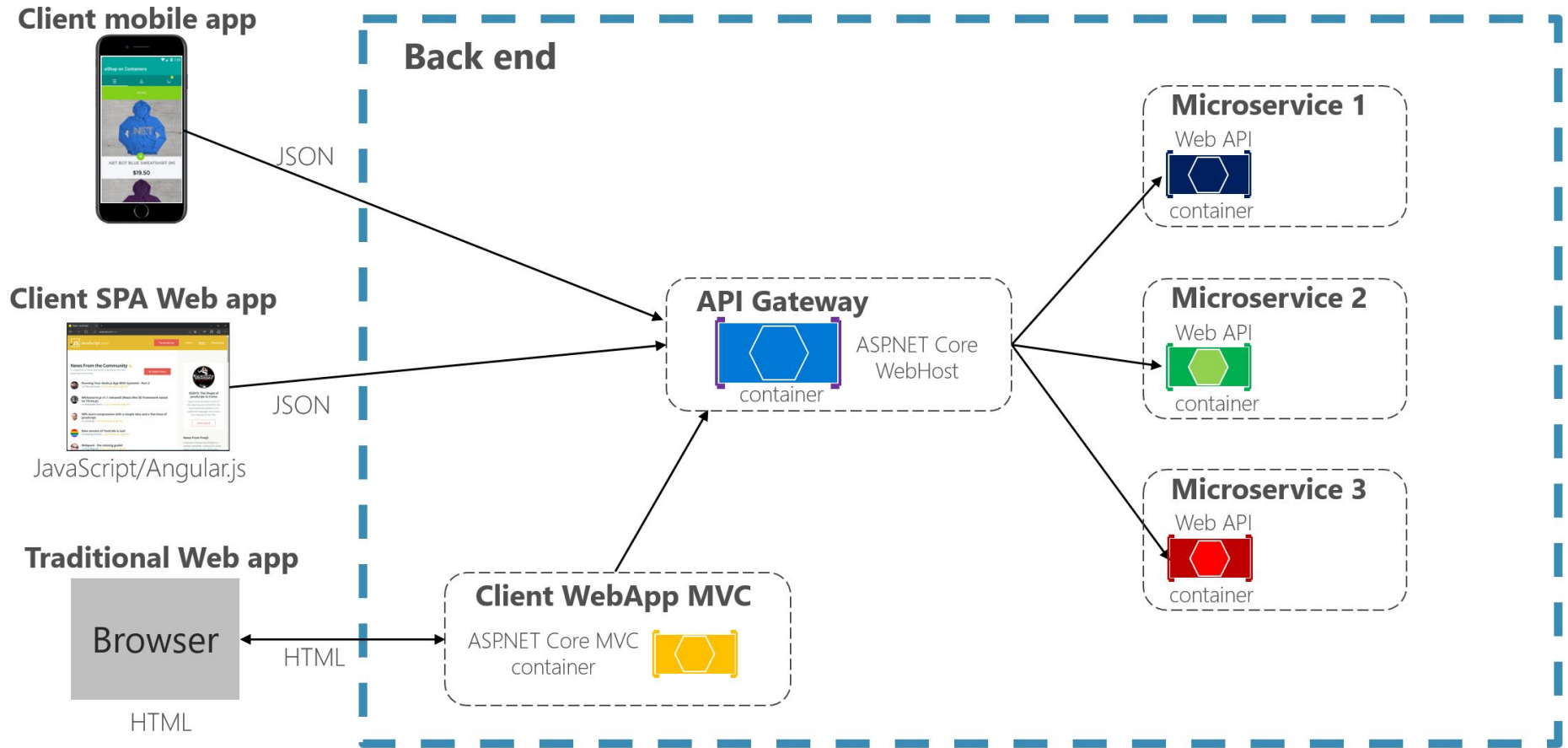


Microservices Architecture

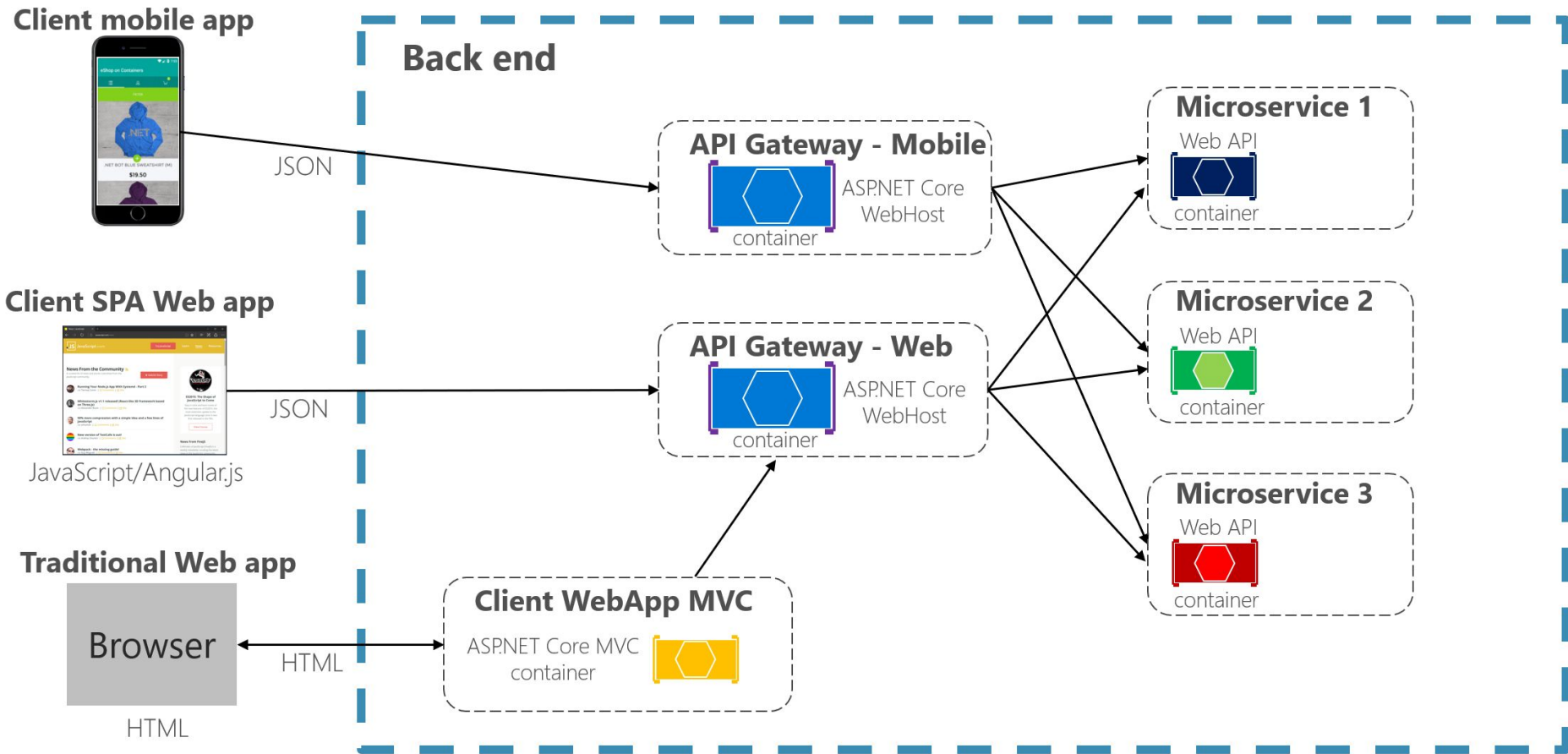
Direct Client-To-Microservice communication Architecture



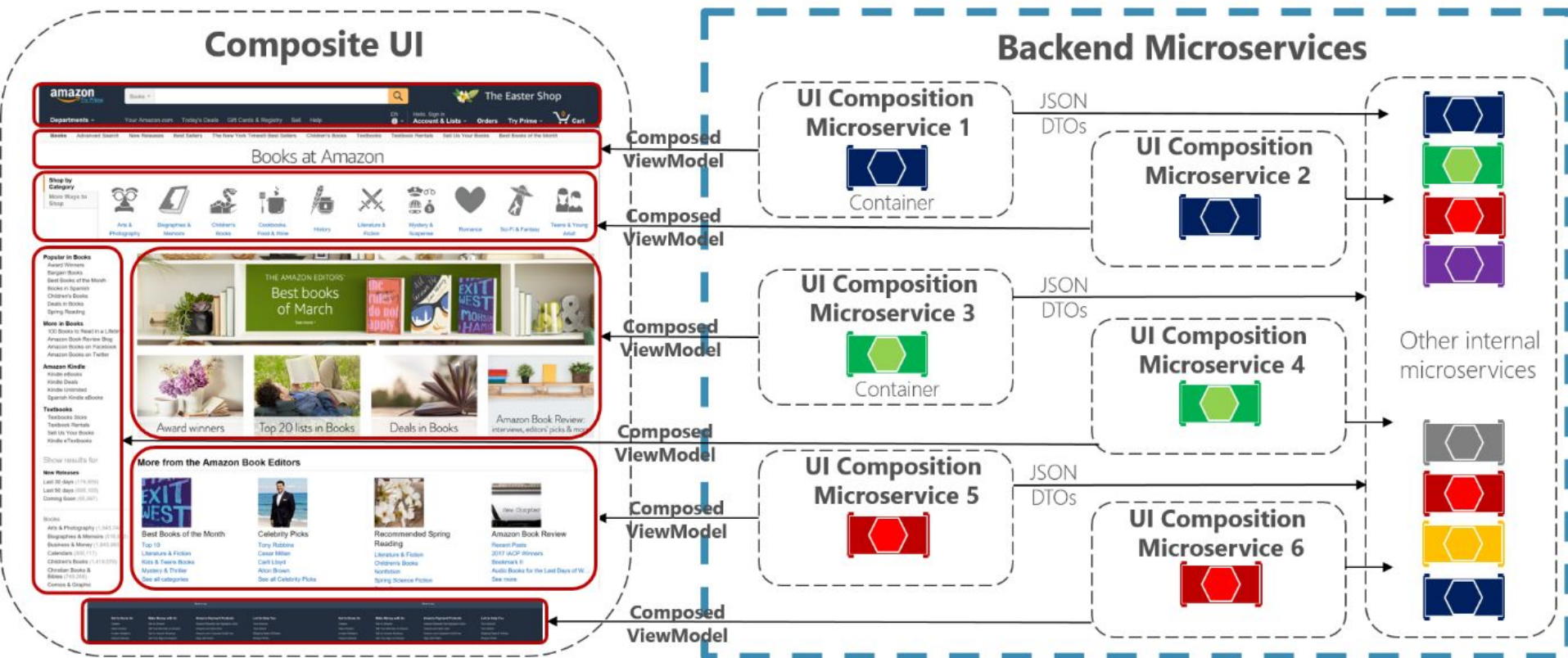
Using a single custom **API Gateway service**

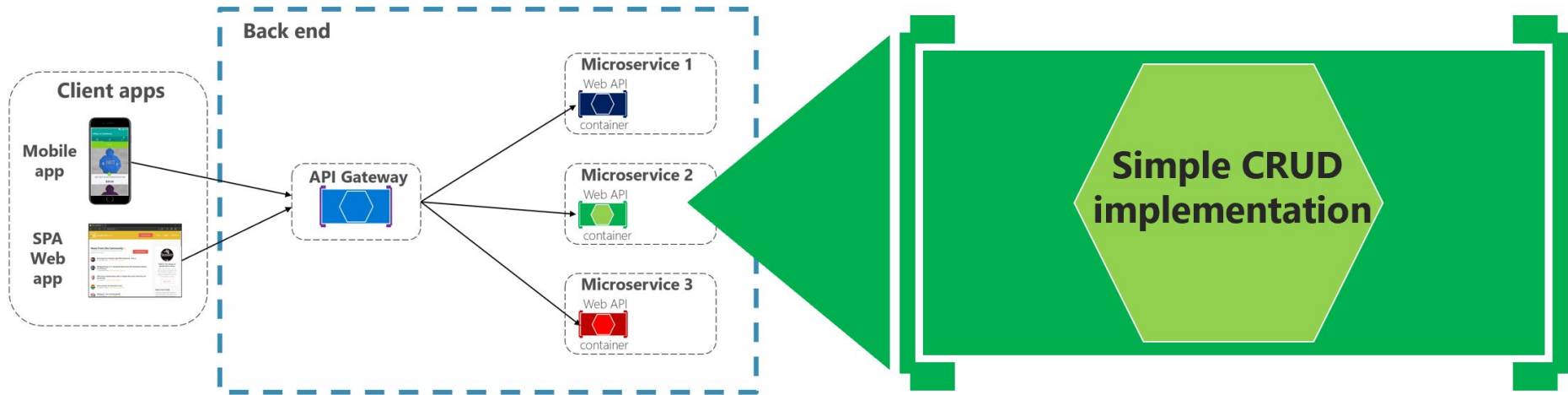


Using multiple **API Gateways / BFF**



Composite UI *generated* by microservices

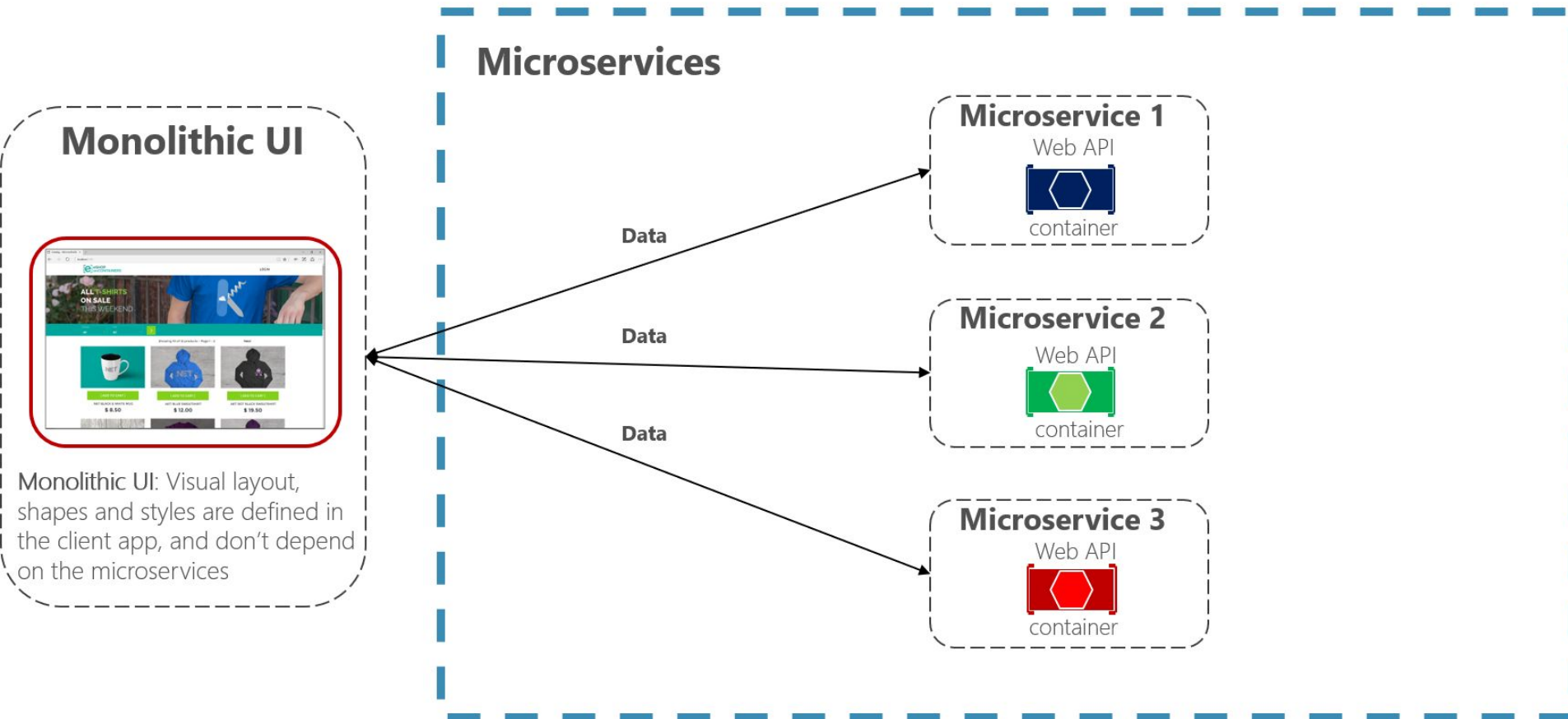




- External microservice patterns
- API Gateway
- Resilient communication
- Pub/Sub and event driven

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).

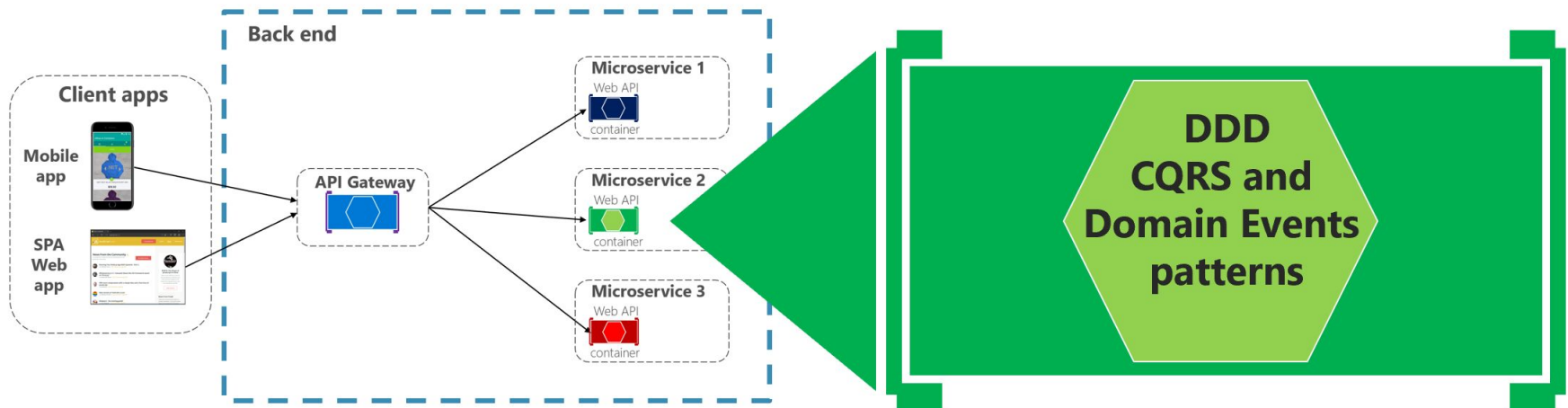
Команды VS запросы: строгое распределение ролей

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

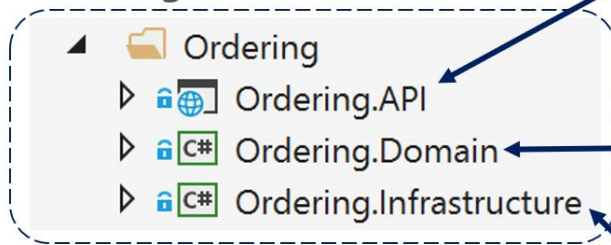


- External microservice patterns
- API Gateway
- Resilient communication
- Pub/Sub and event driven

Internal DDD patterns in addition to SOLID principles and Dependency Injection

Layers in a Domain-Driven Design Microservice

Ordering microservice



Application layer

Domain model layer

Infrastructure layer

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

- 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

- 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.