PROGRAMSKO INŽENJERSTVO

Arhitekturni stilovi

Suvremeni pristupi oblikovanju softverskih sustava

Java / Spring | .NET Core | Python | Next.js

Sadržaj

01

Klijent - Poslužitelj

Temeljna arhitektura današnjeg interneta

Klijent - Poslužitelj: Pregled

Poslužitelji (Server)

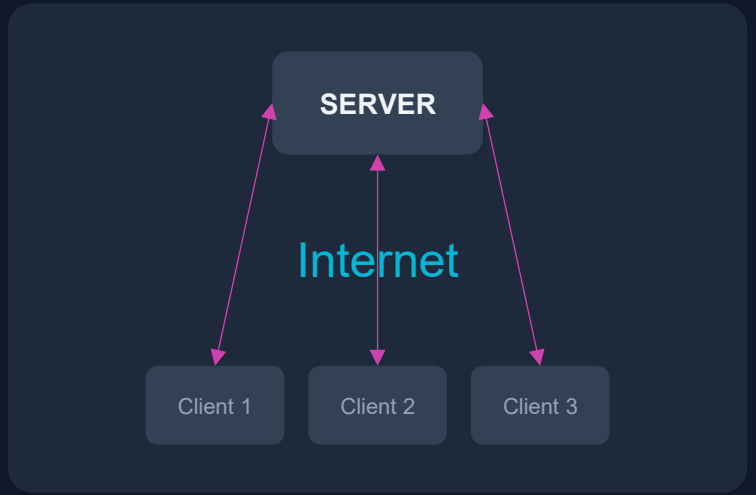
Centri za razmjenu informacija i pružanje usluga. Mogu biti fizička računala ili programi (npr. Apache Tomcat).

Klijenti (Client)

Pristup informacijama i dobivanje usluga. Više klijenata pristupa jednom poslužitelju.

Ograničenja

Broj istovremenih klijenata, duljina reda čekanja, vrijeme trajanja sesije - postavlja administrator.



Tanki vs Debeli klijent

T Tanki klijent (Thin)

Aplikacija ne obavlja zahtjevne proračune niti pohranu podataka.

PRIMJERI:

Web preglednici (Chrome, Firefox)

Cloud aplikacije (Google Docs)

Streaming servisi (Netflix, Spotify)

D Debeli klijent (Fat)

Klijent obavlja većinu obrade zahtjeva. Server služi za interakciju i dijeljenje podataka.

PRIMJERI:

Računalne igre (Steam, LOL)

Desktop aplikacije (Photoshop)

IDE alati (IntelliJ IDEA, VS Code)

Prednosti i Rizici

PREDNOSTI

Posao se može raspodijeliti na više računala.

Klijenti udaljeno pristupaju funkcionalnostima.

Klijent i poslužitelj se oblikuju odvojeno.

Svi podaci mogu se držati na jednom mjestu.

Poslužitelju moze istodobno pristupiti više klijenata.

RIZICI

Sigurnost - enkripcija, vatrozidovi, ovjera korisnika.

Potreba za adaptivnim održavanjem.

Kompatibilnost prema unatrag i unaprijed.

VRSTE POSLUŽITELJA:

HTTP, SMTP, IMAP, FTP, DNS...

02

P2P Arhitektura

Peer-to-Peer - Ravnopravni sudionici

P2P: Ključne karakteristike

Ravnopravni sudionici

Podjela poslova i podataka izmedu računala s jednakim privilegijama i sličnim karakteristikama.

Dijeljenje resursa

Svaki peer ostavlja dio svojih resursa (procesnu moć, prostor na disku, mrežnu propusnost) dostupnu drugim peerovima.

Virtual Overlay Network

Sloj iznad TCP/IP - na razini aplikacijskog sloja peerovi međusobno komuniciraju direktno, neovisno o fizičkoj topologiji.

BitTorrent | Blockchain | IPFS

MESH TOPOLOGY

P2P: Prednosti i Rizici

PREDNOSTI

Nema jedne kritične tocke (*single point of failure*)

Visoka skalabilnost

Peerovi se mogu priduživati i napuštati mrežu transparentno

Ekonomski isplativije - praktično nema troškova održavanja

Decentralizirano upravljanje

RIZICI

Sigurnosni problemi - širenje virusa

Teško nadzirati (može biti i prednost)

Mogućnost širenja nelegalnog sadržaja

Performanse ovise o broju aktivnih peerova

Najbolje za poslove koji se mogu podijeliti u male dijelove nad kojima se radi istovremeno

Višeslojna arhitektura

Layered Architecture

Višeslojna arhitektura

Logička organizacija programske potpore u slojeve

Sloj (Layer)

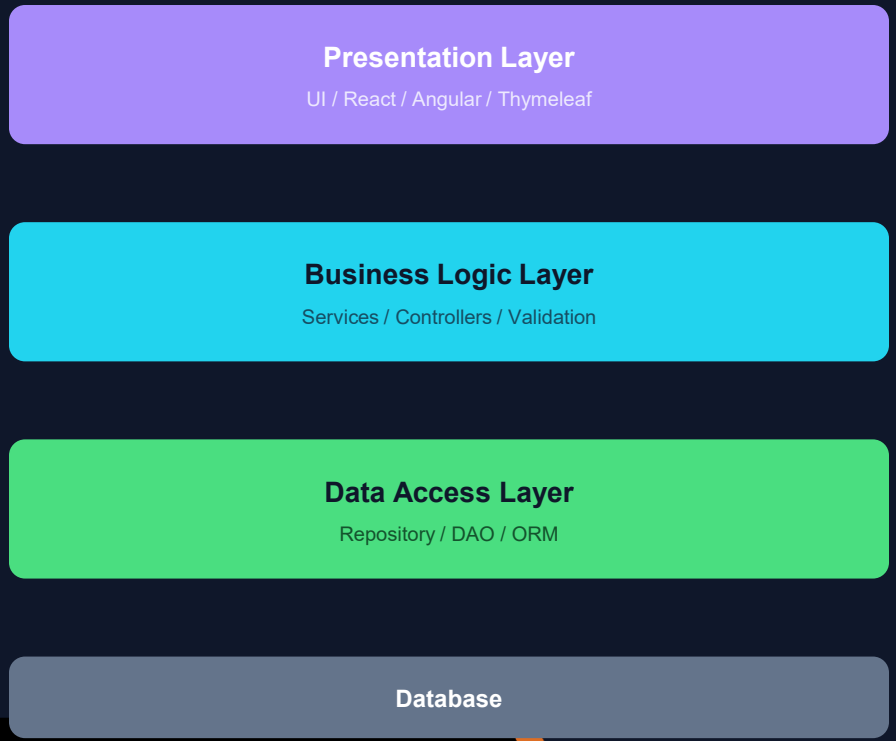
Svaki sloj pruža uslugu drugom sloju i sakriva (enkapsulira) implementacijske detalje

Troslojni stil:

Prezentacijski sloj – korisničko sučelje

Sloj poslovne logike – implementacija procesa i izračuna

Podatkovni sloj – pohrana u bazu ili datotečni sustav



Spring Boot - Slojevita arhitektura

@RestController

```
@RestController
@RequestMapping "/api/users"
public class UserController {

  @Autowired
  private UserService service;


  @GetMapping "/{id}"
  public User get(@PathVariable Long id) {
    return service findById id);
  }
}
```

@Service

```
@Service
public class UserService {

  @Autowired
  private UserRepository repo;

  public User findById(Long id) {
    return repo.findById(id)orElseThrow();
}
```

@Repository

```
@Repository

public interface UserRepository
  extends JpaRepository<User, Long> {
    // Automatic CRUD
    // + custom queries:
    List<User> findByEmail(String email);
}
```

Dependency Injection via @Autowired

Inversion of Control (IoC)

.NET Core - Slojevita arhitektura

[ApiController] - Controller Layer

```
ApiController
Route("api/[controller]" ]
public class UsersController : ControllerBase {
  private readonly IUserService _service;
  public UsersController(IUserService service) => _service = service;

[ HttpGet("{id}")]
  public async Task<ActionResult<User>> Get(int id) => await _service GetByIdAsync id  }
```

Service Layer + DI Registration

```
// IUserService.cs
public interface IUserService {Task<User> GetByIdAsync int id);

// Program.cs (DI Container)
builder.Services.AddScoped<IUserService, UserService>()
                .AddScoped<IUserRepository, UserRepository>()
                .AddDbContext<AppDbContext>();
```

Entity Framework Core - Data Layer

```
public class AppDbContext : DbContext {

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

Višeslojna arhitektura: Evaluacija

✓ PREDNOSTI

- Oblikovanje na temelju više razine apstrakcije
- Jednostavna skalabilnost sustava
- Promjene u sloju utječu samo na okolne slojeve
- Timovi se mogu fokusirati na razvoj zasebnih slojeva
- Podržana je ponovna uporaba (reusability)
- Lakše testiranje - mock slojevi

✗ NEDOSTACI

- Teško odrediti optimalno preslikavanje odgovornosti na slojeve
- Ponekad se funkcionalnosti ne mogu razbiti u slojeve
- Razvoj ovisi o većem broju različitih tehnologija
- Mogući performance overhead zbog prolaska kroz slojeve
- Kompleksnost u komunikaciji među slojevima

Spring Boot | .NET Core | Next.js | Django

Uslužna arhitektura

Service-Oriented Architecture (SOA)

Uslužna arhitektura (SOA)

Programska potpora = kolekcija usluga koje međusobno komuniciraju

Tri temeljna entiteta:

🏭 Pružatelj usluge (Provider)

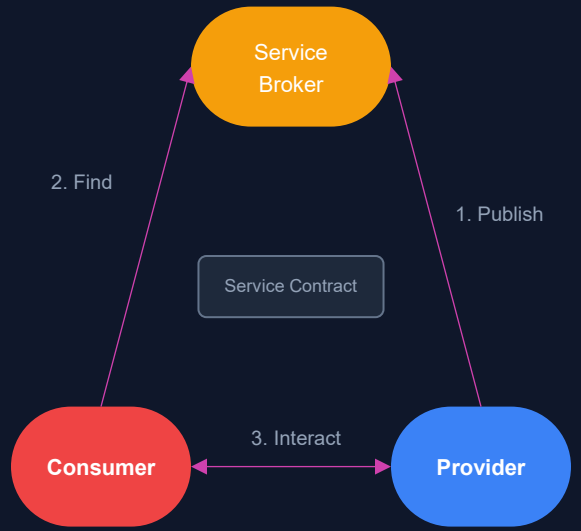
Stvara uslugu i oglašava informacije o njoj u registru

🔍 Posrednik (Broker/Registry)

Služi za otkrivanje informacija o postojanju usluge klijentu

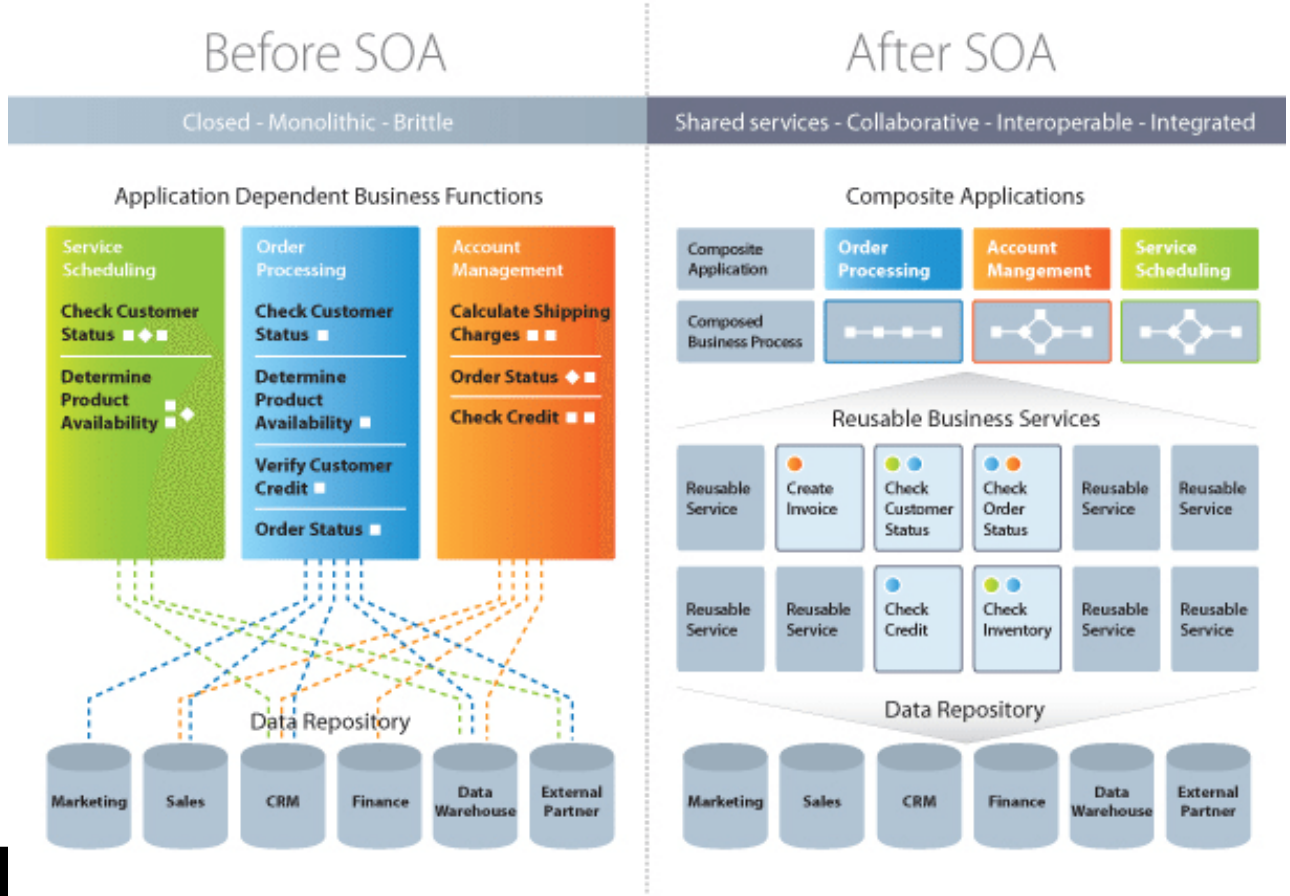
🖩 Klijent (Consumer)

Pronalazi podatke o usluzi u registru i povezuje se s pružateljem



Uslužna arhitektura (SOA)

Prije i poslije



Web usluge

REST vs SOAP + *Modern Alternatives*

REST vs SOAP

REST

Representational State Transfer

Arhitekturni stil, koristi HTTP, XML, JSON, URI

✓ Ne pamti stanje (stateless)
✓ HTTP metode: GET, POST, PUT, DELETE
✓ Lakši za implementaciju
✓ JSON format (lakši od XML)
✓ Cacheable responses

```
// RESTful request
GET /api/users/123
Accept: application/json
```

SOAP

Simple Object Access Protocol

Protokol zasnovan na XML-u, neovisan o platformi

✓ WSDL opis usluge
✓ WS-Security standardi
✓ ACID transakcije
✓ Prijenos preko HTTP, SMTP, FTP...
✗ Veći overhead (XML envelope)

```
// SOAP envelope
<soap:Envelope>
<soap:Body>...</soap:Body>
</soap:Envelope>
```

2025: REST dominira za web API-je, SOAP koristi se u enterprise sustavima s jakim sigurnosnim zahtjevima (bankarstvo, zdravstvo)

2025

Moderne alternative: GraphQL & gRPC

GraphQL

Query language for APIs (Meta, 2015)

✓ Klijent specificira točno koje podatke treba

✓ Jedan endpoint za sve upite

✓ Snažno tipiziran (schema)

```
query {
  user(id: "123") {
    name
    email
    posts {
      title
    }
  }
}
```

Apollo | Relay

gRPC

High-performance RPC (Google)

✓ Protocol Buffers (binarni format)

✓ HTTP/2 + bidirekcijski streaming

✓ Generiranje koda za više jezika

```
// user.proto
service UserService {
  rpc GetUser(UserRequest)
  returns  User
}
```

Microservices | IoT / Mobile

GraphQL: Fleksibilni frontendi, mobilne aplikacije

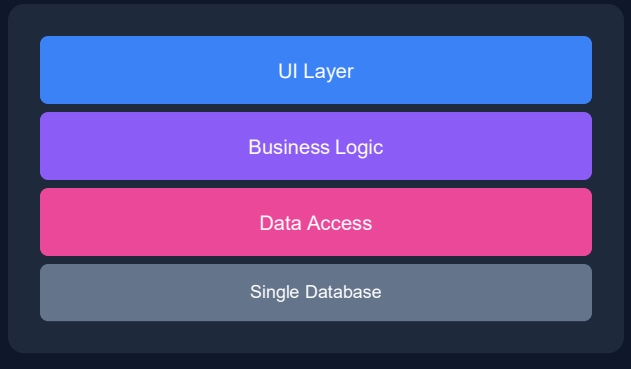
gRPC: Microservice komunikacija, real-time

Mikrousluge

Microservices Architecture

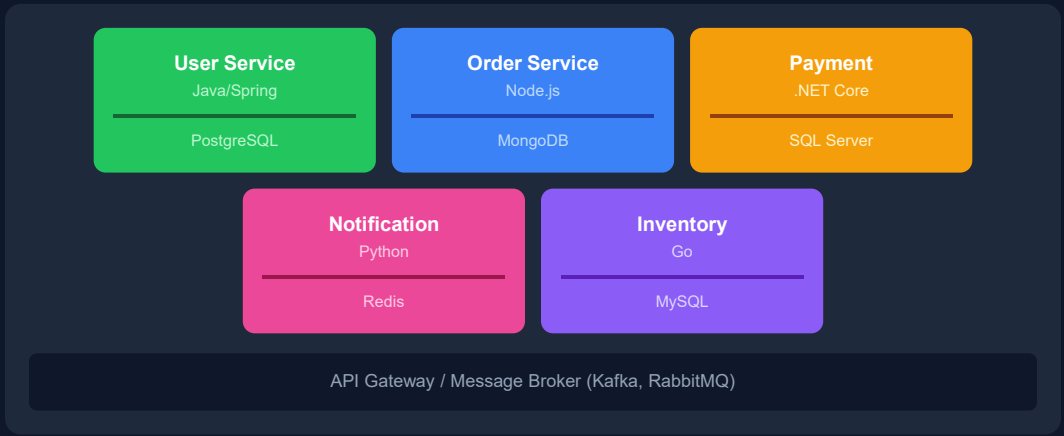
Mikrousluge vs Monolit

MONOLITNA APLIKACIJA



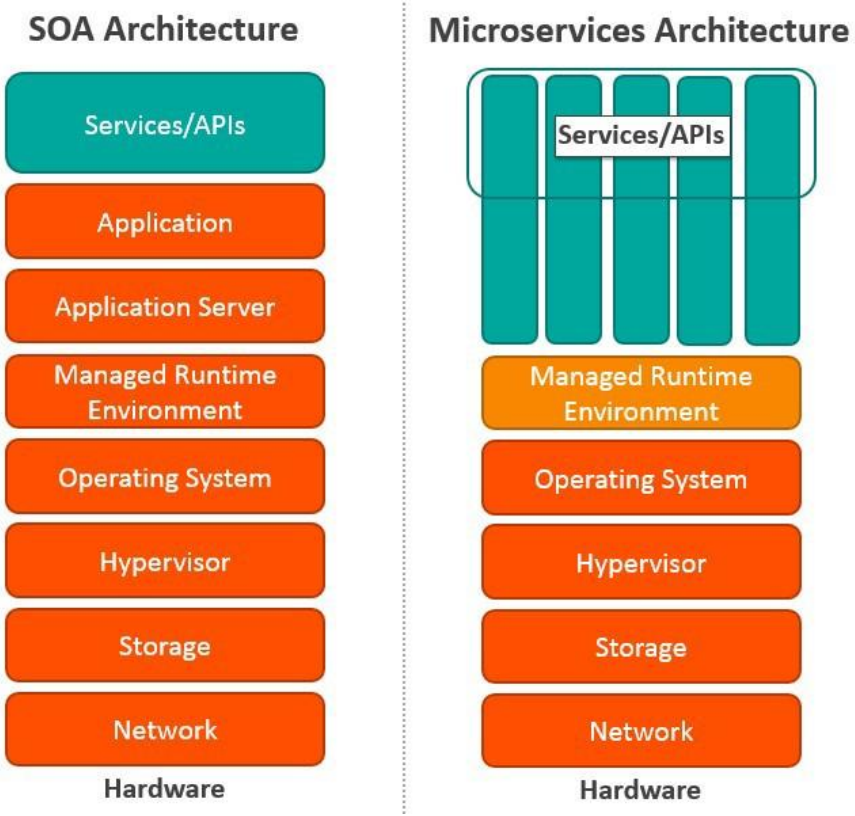
✗ Teško skaliranje
✗ Jedan deploy = cijeli sustav

MIKROUSLUŽNA ARHITEKTURA



✓ Neovisno skaliranje • ✓ Polyglot arhitektura • ✓ Autonomni timovi

SOA arhitektura vs Mikroservisna arhitektura



Arhitektura temeljena na događajima

Event-Driven Architecture (EDA)

Event-Driven Architecture

Koristi događaje za komunikaciju među uslugama - često uz mikrousluge

Publish/Subscribe Pattern

Produceri objavljuju događaje, consumeri se pretplaćuju na teme od interesa

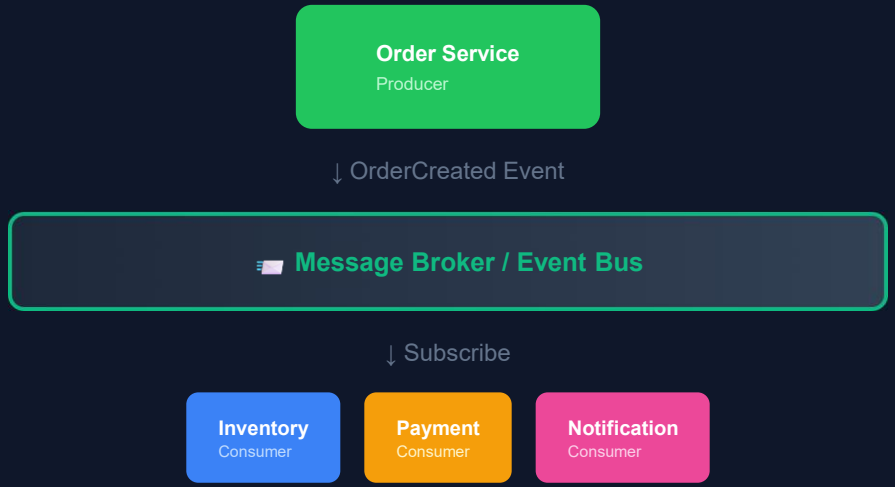
Event Sourcing

Stanje aplikacije rekonstruira se iz slijeda događaja

SAGA Pattern

Upravljanje distribuiranim transakcijama kroz slijed lokalnih transakcija s kompenzacijskim akcijama

Apache Kafka | **RabbitMQ** | **AWS SNS/SQS**



2025

Serverless Architecture

Function as a Service (FaaS)

Kod se izvršava kao reakcija na događaje. Nema upravljanja serverima.

Pay-per-execution

Plaća se samo vrijeme izvršavanja. Idealno za varijabilno opterećenje.

Auto-scaling

Automatsko skaliranje od 0 do tisuća instanci u sekundi.

AWS Lambda | Cloud Functions | Azure Func

```
// AWS Lambda (Node.js)
export const handler = async (event) => {
  const { userId } = event.pathParameters;
  const user = await db.getUser(userId);
  return { statusCode: 200, body: JSON.stringify(user) };
};
```

```
// Azure Functions (Python)
import azure.functions as func

def main(req: func.HttpRequest) -> func.HttpResponse:
    user_id = req.route_params.get('id')
    return func.HttpResponse(user_data)
```

⚠ **Cold Start:** Prvi poziv može biti spor (100ms - 2s)

SWOT analiza arhitekturalnih stilova

Client-Server				
	S - Snage	**W - Slabosti**	**O - Prilike**	**T - Prijetnje**
1.	Centralizirana kontrola i upravljanje	Single point of failure (server)	Cloud migracija za elastičnost	DDoS napadi na centralni server
2.	Jednostavno održavanje i ažuriranje	Visoki troškovi servera	Implementacija load balancera	Zastarijevanje tehnologije
3.	Visoka sigurnost podataka na serveru	Mrežna ovisnost	Hibridni modeli (edge computing)	Konkurencija distribuiranih sustava
4.	Skalabilnost servera prema potrebi	Ograničena horizontalna skalabilnost	Containerizacija servisa	Regulatorne promjene (GDPR)

SWOT analiza arhitekturalnih stilova

Peer-to-Peer (P2P)				
	S - Snage	W - Slabosti	O - Prilike	T - Prijetnje
1.	Nema single point of failure	Teško upravljanje i nadzor	Blockchain tehnologije	Pravne regulacije (piratstvo)
2.	Prirodna skalabilnost	Nekonzistentnost podataka	Decentralizirane aplikacije (dApps)	Malware distribucija
3.	Niski troškovi infrastrukture	Sigurnosni izazovi	Edge computing integracija	Sybil napadi
4.	Otpornost na ispade	Ovisnost o dostupnosti peera	IoT mreže	Manjak povjerenja korisnika

SWOT analiza arhitekturalnih stilova

Slojevita arhitektura				
	S - Snage	W - Slabosti	O - Prilike	T - Prijetnje
1.	Jasna separacija odgovornosti	Performance overhead (prolaz kroz slojeve)	Modernizacija u microservices	Zastarijevanje u odnosu na moderne pristupe
2.	Lakše testiranje po slojevima	Rigidnost strukture	Domain-Driven Design integracija	Tehnički dug
3.	Ponovno korištenje koda	Monolitna priroda	Clean Architecture principi	Konkurencija agilnijih arhitektura
4.	Jednostavno razumijevanje	Teška horizontalna skalabilnost	Modularizacija za lakšu migraciju	Ograničenja cloud-native razvoja

SWOT analiza arhitekturalnih stilova

SOA (Service-Oriented)				
	S - Snage	W - Slabosti	O - Prilike	T - Prijetnje
1.	Interoperabilnost između sustava	Složenost ESB komponente	Integracija legacy sustava	Zamjena s microservices
2.	Ponovna uporaba servisa	Visoki početni troškovi	API Gateway modernizacija	Kompleksnost održavanja ESB-a
3.	Standardizirani protokoli	Overhead SOAP protokola	Hibridni cloud scenariji	Nedostatak stručnjaka
4.	Neovisnost od platforme	Vendor lock-in rizik	B2B integracije	Visoki licencni troškovi

SWOT analiza arhitekturalnih stilova

Microservices				
	S - Snage	W - Slabosti	O - Prilike	T - Prijetnje
1.	Neovisni deployment servisa	Distribuirana kompleksnost	Kubernetes orkestracija	Over-engineering za male projekte
2.	Tehnološka raznolikost (polyglot)	Mrežna latencija	Service mesh (Istio, Linkerd)	Sigurnosni izazovi (više površina napada)
3.	Horizontalna skalabilnost	Teško debugiranje	Serverless integracija	Nedostatak DevOps kulture
4.	Fault isolation	Operational overhead	AI/ML pipeline integracija	Visoki troškovi monitoringa

SWOT analiza arhitekturalnih stilova

Event-Driven				
	S - Snage	W - Slabosti	O - Prilike	T - Prijetnje
1.	Loose coupling komponenti	Kompleksnost praćenja toka	IoT i streaming podataka	Message broker kao single point of failure
2.	Asinkrona komunikacija	Eventual consistency	CQRS i Event Sourcing	Kompleksnost disaster recovery
3.	Visoka skalabilnost	Teško debugiranje	Real-time analytics	Krivulja učenja za tim
4.	Real-time obrada podataka	Dupliciranje i redoslijed poruka	Reactive programiranje	Nedostatak standardizacije

SWOT analiza arhitekturalnih stilova

Serverless				
	S - Snage	W - Slabosti	O - Prilike	T - Prijetnje
1.	Pay-per-use model (nema troška u idle)	Cold start latencija	Edge computing funkcije	Nepredvidivi troškovi pri visokom prometu
2.	Automatska skalabilnost	Vendor lock-in	AI/ML inference	Ograničenja providera
3.	Nema upravljanja infrastrukturom	Ograničeno vrijeme izvršavanja	Event-driven integracije	Sigurnosni rizici dijeljene infrastrukture
4.	Brži time-to-market	Teško lokalno testiranje	Multi-cloud strategije	Promjene cijena providera

Pitanja?

Hvala na pažnji!

Klijent-Poslužitelj | P2P | Višeslojna | SOA | Mikrousluge | Event-Driven | Serverless