Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №9

«Взаємодія компонентів системи»

Виконав
студент групи ІА-34:
Сльота Максим

Перевірив:
Мягкий М. Ю.

Київ - 2025

Тема: Взаємодія компонентів системи.

Мета роботи: Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Serviceoriented Architecture), та реалізувати в проєктованій системі одну із архітектур.

Вихідні дані:

17. **System activity monitor** (iterator, command, abstract factory, bridge, visitor, SOA)

Монітор активності системи повинен зберігати і запам'ятовувати статистику використовуваних компонентів системи, включаючи навантаження на процесор, обсяг займаної оперативної пам'яті, натискання клавіш на клавіатурі, дії миші (переміщення, натискання), відкриття вікон і зміна вікон; будувати звіти про використання комп'ютера за різними критеріями (% часу перебування у веб-браузері, середнє навантаження на процесор по годинах, середній час роботи комп'ютера по днях і т.д.); правильно поводитися з «простоюванням» системи – відсутністю користувача.

Теоретичні відомості:

Клієнт-серверна архітектура - Клієнт-серверні додатки являють собою найпростіший варіант розподілених додатків, де виділяється два види додатків: клієнти (представляють додаток користувачеві) і сервери (використовується для зберігання і обробки даних). Розрізняють тонкі клієнти і товсті клієнти. Тонкий клієнт – клієнт, який повністю всі операції (або більшість, пов'язаних з логікою роботи програми) передає для обробки на сервер, а сам зберігає лише візуальне уявлення одержуваних від сервера відповідей. Грубо кажучи, тонкий клієнт – набір форм відображення і канал зв'язку з сервером. Прикладом тонкого клієнта є класичні Web-застосунки.

Peer-to-Peer архітектура - це модель мережевої взаємодії, в якій кожен вузол (комп'ютер або пристрій) є одночасно клієнтом і сервером. У цій архітектурі всі вузли мають рівні права та можливості для обміну даними, ресурсами або виконання завдань. На відміну від клієнт-серверної моделі, де є чітке розділення на клієнти й сервери, P2P-мережа дозволяє учасникам взаємодіяти безпосередньо, без необхідності в централізованому сервері.

Сервіс-орієнтована архітектура - модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабо пов'язаних (англ. Loose coupling) сервісів або служб, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами. Історично сервіс-орієнтована архітектура появилася як альтернатива монолітній архітектурі, в які вся система розроблялася та розгорталася як одне ціле.

Мікро-сервісна архітектура - є підходом до створення серверного додатку як набору малих служб [11]. Це означає, що архітектура мікро-сервісів головним чином орієнтована на серверну частину, не дивлячись на те, що цей підхід так само використовується для зовнішнього інтерфейсу, де кожна служба виконується в своєму процесі і взаємодіє з іншими службами за такими протоколами, як HTTP/HTTPS, WebSockets чи AMQP.

Хід роботи:

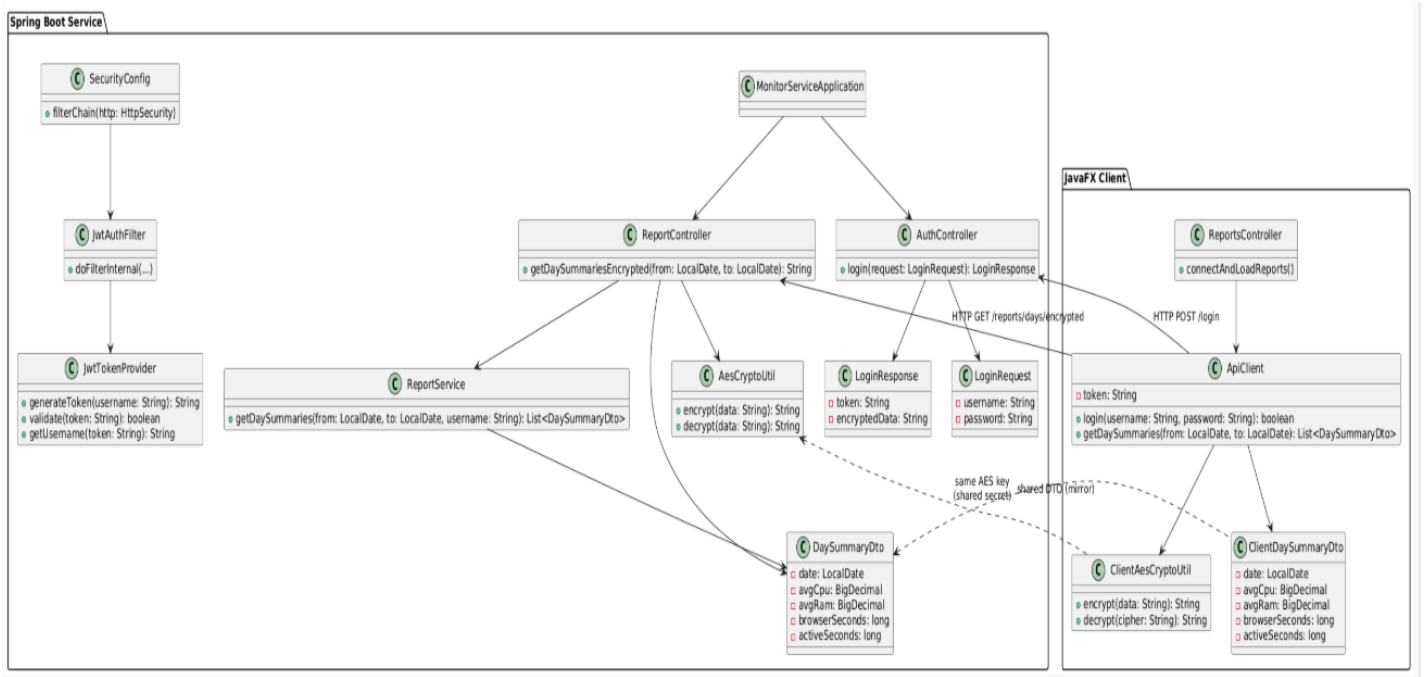

Рис 1. Діаграма класів

У діаграмі зображено архітектуру розподіленої системи, у якій серверна частина, реалізована на базі Spring Boot, взаємодіє з клієнтським застосунком JavaFX у форматі сервіс-орієнтованої архітектури. У центральній ролі на сервері знаходиться веб-сервіс, що забезпечує автентифікацію користувача, перевірку токенів і надання статистичних даних у зашифрованому вигляді. Клас SecurityConfig визначає правила безпеки, вмикає JWT-фільтр і контролює доступ до REST-ендпойнтів. JwtAuthFilter перехоплює кожен запит, перевіряє валідність JWT-токена та встановлює дані

автентифікації у контекст безпеки. Генерація і валідація токенів здійснюється через JwtTokenProvider, який формує підписані маркери доступу з терміном дії. Контролер AuthController відповідає за вхід користувача: він приймає модель LoginRequest, перевіряє облікові дані та повертає LoginResponse із JWT-токеном і додатковим AES-шифрованим повідомленням, яке підтверджує успішну автентифікацію.

Другий серверний контролер, ReportController, забезпечує отримання статистики роботи системи. Після успішної перевірки токена він звертається до ReportService, який формує набір об’єктів DaySummaryDto, що описують середнє навантаження на процесор, пам’ять, активний час користувача та активність у браузері. Перед відправленням дані шифруються симетричним ключем за допомогою AesCryptoUtil, щоб клієнт отримував інформацію у захищеному вигляді. Усі моделі LoginRequest, LoginResponse та DaySummaryDto використовуються як транспортні об’єкти між сервером і клієнтом.

У правій частині діаграми представлений клієнтський застосунок. Його основним елементом є ApiClient, який відправляє HTTP-запити на сервер, отримує JWT, зберігає його й додає до всіх наступних запитів. Після отримання статистики ApiClient виконує розшифрування даних тим самим AES-ключем, що використовується на сервері. Клас ClientAesCryptoUtil є повною клієнтською копією серверного AesCryptoUtil, що дозволяє виконувати симетричне шифрування та дешифрування. Розшифровані результати перетворюються у моделі ClientDaySummaryDto і передаються у ReportsController, який відповідає за відображення статистики у графічному інтерфейсі JavaFX.

Діаграма демонструє цілісну SOA-архітектуру, де сервер виконує роль постачальника сервісів, а клієнт — їх споживача. Взаємодія між ними побудована на використанні JWT-токенів для автентифікації та AES-шифрування для захисту даних. Клієнт і сервер працюють як ізольовані компоненти, обмінюючись лише стандартизованими запитами та DTO-об’єктами, що робить рішення масштабованим, безпечним і придатним до подальшого розширення.

Доданий вихідний код системи:

MonitorServiceApplication.java

```
@SpringBootApplication
public class MonitorServiceApplication {
   public static void main(String[] args) {
```

```
        SpringApplication.run(MonitorServiceApplication.class, args);
    }
}
```

AesCryptoUtil.java

```
public class AesCryptoUtil {
    private static final byte[] IV = "0123456789ABCDEF".getBytes();
  private static final String SECRET = "SuperSecretKeyForAES256Monitor!!";
    private static SecretKeySpec getKey() {
        byte[] keyBytes = SECRET.getBytes();
        byte[] key = new byte[32];
        System.arraycopy(keyBytes, 0, key, 0, Math.min(keyBytes.length, 32));
        return new SecretKeySpec(key, "AES");
    }
    public static String encrypt(String plainText) {
        try {
            Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
            GCMParameterSpec spec = new GCMParameterSpec(128, IV);
            cipher.init(Cipher.ENCRYPT_MODE, getKey(), spec);
            byte[] enc = cipher.doFinal(plainText.getBytes());
            return Base64.getEncoder().encodeToString(enc);
        } catch (Exception e) {
            throw new RuntimeException("AES encrypt error", e);
        }
    }
    public static String decrypt(String cipherText) {
        try {
            byte[] decoded = Base64.getDecoder().decode(cipherText);
            Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
            GCMParameterSpec spec = new GCMParameterSpec(128, IV);
            cipher.init(Cipher.DECRYPT_MODE, getKey(), spec);
            byte[] dec = cipher.doFinal(decoded);
            return new String(dec);
```

```
        } catch (Exception e) {
            throw new RuntimeException("AES decrypt error", e);
        }
    }
}
```

JwtTokenProvider.java

```
public class JwtTokenProvider {
    private static final String JWT_SECRET =
"SuperJwtSecretForSystemActivityMonitor2024!";
    private static final long EXPIRATION_MS = 3600_000; // 1 година
    private static Key getSigningKey() {
        byte[] keyBytes = JWT_SECRET.getBytes();
        return Keys.hmacShaKeyFor(keyBytes);
    }
    public static String generateToken(String username) {
        Date now = new Date();
        Date exp = new Date(now.getTime() + EXPIRATION_MS);

        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(now)
                .setExpiration(exp)
                .signWith(getSigningKey(), SignatureAlgorithm.HS256)
                .compact();
    }
    public static String getUsername(String token) {
        return Jwts.parserBuilder()
                .setSigningKey(getSigningKey())
                .build()
                .parseClaimsJws(token)
                .getBody()
                .getSubject();
```

```
    }

    public static boolean validate(String token) {
        try {
            Jwts.parserBuilder()
                    .setSigningKey(getSigningKey())
                    .build()
                    .parseClaimsJws(token);
            return true;
        } catch (JwtException | IllegalArgumentException e) {
            return false;
        }
    }
}
```

JwtAuthFilter.java

```
public class JwtAuthFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain chain)
            throws ServletException, IOException {

        String header = request.getHeader(HttpHeaders.AUTHORIZATION);
        if (header != null && header.startsWith("Bearer ")) {
            String token = header.substring(7);
            if (JwtTokenProvider.validate(token)) {
                String username = JwtTokenProvider.getUsername(token);
                var auth = new UsernamePasswordAuthenticationToken(
                        username,
                        null,
                        Collections.emptyList()
```

```
                );
                auth.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(auth);
            }
        }
        chain.doFilter(request, response);
    }
}
@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        http.csrf(csrf -> csrf.disable());

        http.authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .anyRequest().authenticated()
        );

        http.addFilterBefore(new JwtAuthFilter(),
UsernamePasswordAuthenticationFilter.class);

        http.httpBasic(Customizer.withDefaults());

        return http.build();
    }
}
LoginRequest.java
public class LoginRequest {
    private String username;
```

```
    private String password;

    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }

    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
}
```

LoginResponse.java

```
public class LoginResponse {
    private String token;
    private String encryptedData;
    public LoginResponse() {}
    public LoginResponse(String token, String encryptedData) {
        this.token = token;
        this.encryptedData = encryptedData;
    }
    public String getToken() { return token; }
    public void setToken(String token) { this.token = token; }
    public String getEncryptedData() { return encryptedData; }
    public void setEncryptedData(String encryptedData) { this.encryptedData =
encryptedData; }
}
```

DaySummaryDto.java

```
public class DaySummaryDto {
    private LocalDate date;
    private BigDecimal avgCpu;
    private BigDecimal avgRam;
    private long browserSeconds;
    private long activeSeconds;
    public LocalDate getDate() { return date; }
    public void setDate(LocalDate date) { this.date = date; }
```

```
    public BigDecimal getAvgCpu() { return avgCpu; }
    public void setAvgCpu(BigDecimal avgCpu) { this.avgCpu = avgCpu; }
    public BigDecimal getAvgRam() { return avgRam; }
    public void setAvgRam(BigDecimal avgRam) { this.avgRam = avgRam; }
    public long getBrowserSeconds() { return browserSeconds; }
    public void setBrowserSeconds(long browserSeconds) { this.browserSeconds =
browserSeconds; }
    public long getActiveSeconds() { return activeSeconds; }
    public void setActiveSeconds(long activeSeconds) { this.activeSeconds = activeSeconds;
}
}
```

ReportService.java

```
public class ReportService {
    public List<DaySummaryDto> getDaySummaries(LocalDate from, LocalDate to, String
username) {
        List<DaySummaryDto> list = new ArrayList<>();
        LocalDate d = from;
        while (!d.isAfter(to)) {
            DaySummaryDto dto = new DaySummaryDto();
            dto.setDate(d);
            dto.setAvgCpu(new BigDecimal("35.5"));
            dto.setAvgRam(new BigDecimal("48.3"));
            dto.setBrowserSeconds(3600);
            dto.setActiveSeconds(7 * 3600);
            list.add(dto);
            d = d.plusDays(1);
        }
        return list;
    }
}
```

AuthController.java

```
@RestController
```

```
@RequestMapping("/api/auth")
public class AuthController {
 @PostMapping("/login")
   public ResponseEntity<LoginResponse> login(@RequestBody LoginRequest request) {

      if ("admin".equals(request.getUsername()) &&
         "admin".equals(request.getPassword())) {
         String token = JwtTokenProvider.generateToken(request.getUsername());
         String encryptedInfo = AesCryptoUtil.encrypt("AUTH_OK");

         return ResponseEntity.ok(new LoginResponse(token, encryptedInfo));
      }
      return ResponseEntity.status(401).build();
   }
}
```

ReportController.java

```
@RestController
@RequestMapping("/api/reports")
public class ReportController {

   private final ReportService reportService = new ReportService();
   private final ObjectMapper objectMapper = new ObjectMapper();

   @GetMapping("/days/encrypted")
   public ResponseEntity<String> getDaySummariesEncrypted(
         Authentication auth,
         @RequestParam("from") @DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
LocalDate from,
         @RequestParam("to")   @DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
LocalDate to
   ) throws Exception {
```

```
        String username = (String) auth.getPrincipal();
        List<DaySummaryDto> list = reportService.getDaySummaries(from, to, username);

        String json = objectMapper.writeValueAsString(list);

        String encrypted = AesCryptoUtil.encrypt(json);

        return ResponseEntity.ok(encrypted);
    }
}
```

AesCryptoUtil.java

```
public class AesCryptoUtil {

    private static final byte[] IV = "0123456789ABCDEF".getBytes();
    private static final String SECRET = "SuperSecretKeyForAES256Monitor!!";

    private static SecretKeySpec getKey() {
        byte[] keyBytes = SECRET.getBytes();
        byte[] key = new byte[32];
        System.arraycopy(keyBytes, 0, key, 0, Math.min(keyBytes.length, 32));
        return new SecretKeySpec(key, "AES");
    }

    public static String encrypt(String plainText) {
        try {
            Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
            GCMParameterSpec spec = new GCMParameterSpec(128, IV);
            cipher.init(Cipher.ENCRYPT_MODE, getKey(), spec);
            byte[] enc = cipher.doFinal(plainText.getBytes());
            return Base64.getEncoder().encodeToString(enc);
        } catch (Exception e) {
            throw new RuntimeException("AES encrypt error", e);
```

```
        }
    }
    public static String decrypt(String cipherText) {
        try {
            byte[] decoded = Base64.getDecoder().decode(cipherText);
            Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
            GCMParameterSpec spec = new GCMParameterSpec(128, IV);
            cipher.init(Cipher.DECRYPT_MODE, getKey(), spec);
            byte[] dec = cipher.doFinal(decoded);
            return new String(dec);
        } catch (Exception e) {
            throw new RuntimeException("AES decrypt error", e);
        }
    }
}
```

ApiClient.java

```
public class ApiClient {

    private final HttpClient httpClient = HttpClient.newHttpClient();
    private final ObjectMapper mapper = new ObjectMapper();

    private String baseUrl = "http://localhost:8080";
    private String token; // JWT

    public void setBaseUrl(String baseUrl) {
        this.baseUrl = baseUrl;
    }

    public boolean login(String username, String password) {
        try {
            LoginRequest req = new LoginRequest(username, password);
            String json = mapper.writeValueAsString(req);
```

```
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(baseUrl + "/api/auth/login"))
                .header("Content-Type", "application/json")
                .POST(HttpRequest.BodyPublishers.ofString(json))
                .build();

        HttpResponse<String> response =
                httpClient.send(request, HttpResponse.BodyHandlers.ofString());

        if (response.statusCode() == 200) {
            LoginResponse lr = mapper.readValue(response.body(), LoginResponse.class);
            this.token = lr.getToken();

            // Приклад: розшифрувати службове повідомлення
            String info = AesCryptoUtil.decrypt(lr.getEncryptedData());
            System.out.println("Server auth info (decrypted): " + info);

            return true;
        } else {
            System.out.println("Login failed, status = " + response.statusCode());
            return false;
        }

    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
public List<DaySummaryDto> getDaySummaries(LocalDate from, LocalDate to) {
    if (token == null) {
        throw new IllegalStateException("Not authenticated");
```

```
        }

        try {
            String url = String.format("%s/api/reports/days/encrypted?from=%s&to=%s",
                    baseUrl, from, to);

            HttpRequest request = HttpRequest.newBuilder()
                    .uri(URI.create(url))
                    .header("Authorization", "Bearer " + token)
                    .GET()
                    .build();

            HttpResponse<String> response =
                    httpClient.send(request, HttpResponse.BodyHandlers.ofString());

            if (response.statusCode() == 200) {
                String encrypted = response.body();
                String json = AesCryptoUtil.decrypt(encrypted);

                DaySummaryDto[] arr = mapper.readValue(json, DaySummaryDto[].class);
                return Arrays.asList(arr);
            } else {
                System.out.println("getDaySummaries failed, status = " + response.statusCode());
                return List.of();
            }
        } catch (Exception e) {
            e.printStackTrace();
            return List.of();
        }
    }
}
```

ReportsController.java

```
public class ReportsController {

    @FXML private DatePicker startDatePicker;
    @FXML private DatePicker endDatePicker;
    @FXML private Label messageLabel;
    @FXML private ListView<String> reportsListView;

    private final ApiClient apiClient = new ApiClient();

    @FXML
    private void initialize() {
        // за бажанням: apiClient.setBaseUrl("http://127.0.0.1:8080");
    }

    @FXML
    private void connectAndLoadReports() {
        LocalDate from = startDatePicker.getValue();
        LocalDate to   = endDatePicker.getValue();

        if (from == null || to == null) {
            messageLabel.setText("Оберіть діапазон дат");
            return;
        }

        // краще запускати в окремому потоці
        new Thread(() -> {
            boolean ok = apiClient.login("admin", "admin");
            if (!ok) {
                Platform.runLater(() -> messageLabel.setText("Помилка автентифікації"));
                return;
            }
```

```
            List<DaySummaryDto> list = apiClient.getDaySummaries(from, to);

            Platform.runLater(() -> {
                reportsListView.getItems().clear();
                for (DaySummaryDto dto : list) {
                    String line = String.format("%s | CPU: %s%% | RAM: %s%% | Active: %d сек |
Browser: %d сек",
                            dto.getDate(),
                            dto.getAvgCpu(),
                            dto.getAvgRam(),
                            dto.getActiveSeconds(),
                            dto.getBrowserSeconds());
                    reportsListView.getItems().add(line);
                }
                messageLabel.setText("Дані отримані з SOA-сервісу");
            });

        }).start();
    }
}
```

Висновок: У цій лабораторній роботі було розроблено повноцінну сервіс-орієнтовану архітектуру, у якій клієнтський застосунок та сервер функціонують як окремі, незалежні компоненти, що взаємодіють між собою через стандартизовані Web-сервіси. У ході роботи вдалося реалізувати сервер на базі Spring Boot, який забезпечує прийом і обробку запитів, виконує автентифікацію користувачів за допомогою JWT-токенів та формує захищені відповіді з використанням симетричного AES-шифрування. Це дало можливість не лише гарантувати коректність доступу до ресурсів, але й забезпечити конфіденційність даних під час їх передавання мережею. Клієнтська частина у вигляді JavaFX-застосунку була інтегрована з веб-сервісом через спеціально створений ApiClient, який виконує авторизацію, надсилає запити, отримує зашифровані відповіді та розшифровує їх відповідним AES-ключем. Такий підхід продемонстрував переваги рознесення логіки між окремими сервісами: клієнт не містить бізнес-логіки обробки

статистики, а лише відображає отримані дані, тоді як сервер відповідає за валідацію, безпеку та формування результатів.

Відповідь на контрольні питання:

1. Клієнт-серверна архітектура — це спосіб побудови програми, де є дві частини: клієнт і сервер. Клієнт — це те, чим користується людина (наприклад, програма на ПК чи браузер). Сервер — це “мозок”, який робить обробку, зберігання, логіку, бази даних. Клієнт просить, сервер відповідає.
2. Сервіс-орієнтована архітектура (SOA) — це спосіб побудови системи, де функціональність розділена на окремі сервіси, які працюють як незалежні блоки. Кожен сервіс виконує одну конкретну роботу: авторизація, обробка платежів, логування, аналітика. Сервіси можуть бути написані на різних мовах, працювати на різних серверах і навіть розвиватись різними командами. Головне, що вони спілкуються між собою через стандартизовані протоколи — зазвичай через HTTP і обмін структурованими даними.
3. SOA керується простими принципами. Сервіси мають бути незалежними, повторно використовуваними, з чіткими контрактами (описами того, що вони вміють), і не повинні залежати від внутрішньої реалізації один одного. Важливо, щоб сервіси були доступні по стандартних протоколах, щоб їх можна було знайти, викликати і підключити в будь-яку систему.
4. Сервіси в SOA взаємодіють через мережеві запити. Один сервіс просто робить запит до іншого через HTTP, XML-повідомлення, SOAP або REST. Комунікація виглядає майже як робота з API: відправив дані — отримав результат. Вони не викликають функції напряму, як у звичайному коді, а працюють так, ніби роблять запит “до іншого комп’ютера”.
5. Розробники дізнаються про сервіси через їх описи, документацію або реєстри. У SOA часто є спеціальний “каталог сервісів”, куди кожен сервіс реєструється. Там можна подивитися, які сервіси існують, що вони роблять і як до них звертатися. По суті, це каталог API. Розробник читає опис сервісу, URL, методи, параметри — і робить до нього запит, як до будь-якого API.
6. Клієнт-серверна модель має свої плюси й мінуси. Переваги в тому, що логіка централізована — простіше оновлювати, простіше захищати дані, простіше масштабувати сервер. Клієнт залишається легким і простим. Недоліки в тому, що якщо сервер ляже — ляже все. А ще потрібні ресурси для підтримки сервера, і інколи можуть

бути затримки через мережу.
7. Однорангова модель (P2P) має свої плюси й мінуси. Перевага в тому, що всі вузли рівні: кожен може бути і клієнтом, і сервером. Система не залежить від одного центру, а значить працює навіть якщо частина вузлів відпала. Недолік у тому, що важко забезпечити єдине управління, безпеку й узгодженість — кожен вузол сам по собі, систему складно контролювати і масштабувати під серйозні бізнес-процеси.
8. Мікросервісна архітектура — це еволюція SOA, але ще більш дрібна і незалежна. У мікросервісах кожна невелика частина системи — це окремий сервіс. Вони маленькі, легко замінюються, легко оновлюються, і кожен можна розгортати окремо. По суті, це SOA, але з більш жорстким акцентом на незалежності та автономності кожного сервісу.
9. У мікросервісах працюють ті ж протоколи, що і в SOA, але частіше використовують легкі формати. Найпопулярніше — REST через HTTP і передача JSON. Також використовують gRPC (дуже швидкий варіант), WebSockets, Kafka, RabbitMQ для обміну подіями. SOAP у мікросервісах практично не використовують, він більше зі старого світу SOA.
10. Якщо в проєкті між веб-контролерами й шаром даних є сервісний шар — це ще не SOA. Це просто нормальна багатошарова архітектура (MVC, Layered Architecture). У SOA сервіс — це окремий застосунок, який працює в мережі й має свій API. А сервісний шар у вашій програмі — це внутрішні класи, які не є окремими сервісами. Тобто підхід правильний, але це не SOA, а просто хороша структура проєкту.