

ДОДАТОК А

Слайди презентації

Харківський національний університет радіоелектроніки
Центр післядипломної освіти

Кафедра програмної інженерії

АТЕСТАЦІЙНА РОБОТА
другого (магістерського) рівня вищої освіти

**Дослідження методів розробки
сервіс-орієнтованих додатків для
використання у публічних хмарах**

Виконав: студент 2 курсу групи ІПЗмзд-17-1 спеціальності
121 Інженерія програмного забезпечення **Колосов І.Ю.**

Науковий керівник: **доц.Лановий О.Ф.**

Харків, 2018

Рисунок А.1 – Титульний слайд

Мета роботи:

Об'єкт дослідження – технології розробки сервісного програмного забезпечення

Предмет дослідження – процес розробки сервіс-орієнтованого ПЗ на базі технології мікросервісів

Мета атестаційної роботи – за результатами аналізу предметної області дослідити особливості процесів розробки додатків SOA для організації інтегрованого доступу до сервісів різних хмарних провайдерів

Наукова новизна – недостатньо вивчені методи, що дозволяють в одному додатку одночасно використовувати сервіси різних провайдерів

Рисунок А.2 – Мета роботи

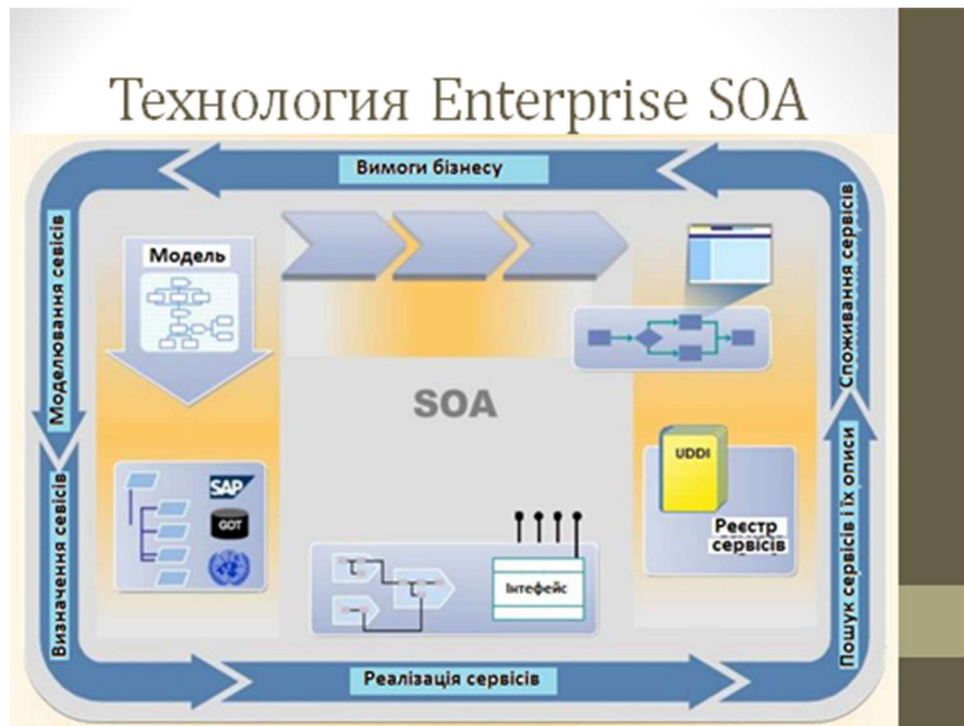


Рисунок А.3 – Технологія Enterprise SOA



Рисунок А.4 – Зв'язок між хмарними сервісами

Постановка задачі дослідження

- побудувати модель
- визначити вимоги та обмеження
- виділити підзадачі
- описати набір модулів та їх функції, які необхідні для розв'язання підзадач
- обрати технологію програмування та виконати розробку програмного забезпечення
- виконати тестування додатку

Рисунок А.5 – Постановка задачі дослідження

Можливі шляхи розв'язання проблеми

- побудувати віртуальну хмару

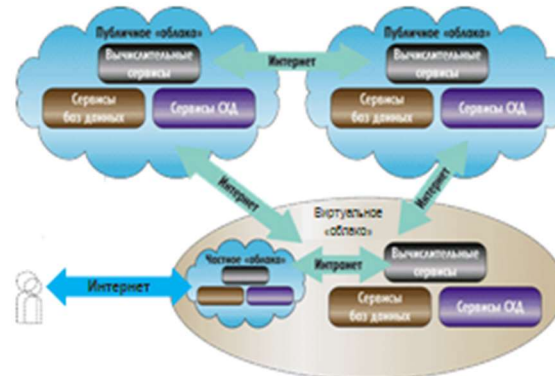


Рисунок А.6 – Шляхи розв'язання проблеми (1)

Можливі шляхи розв'язання проблеми

- побудувати програмний інтегратор

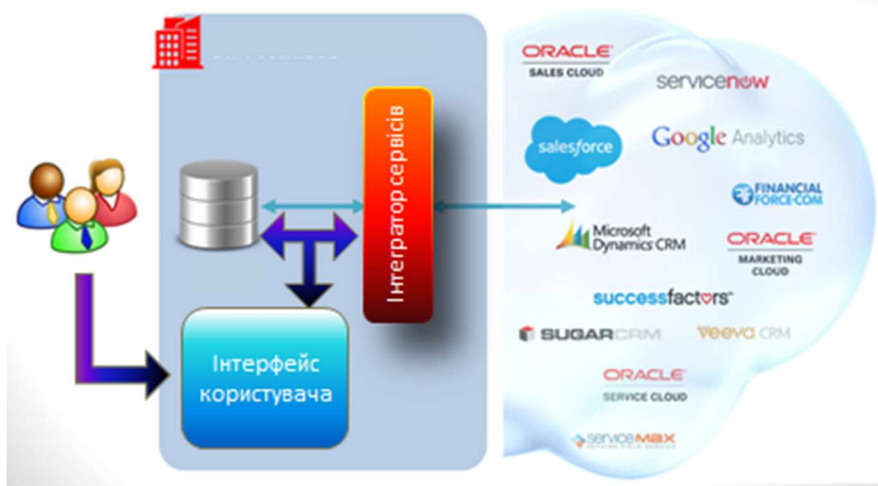


Рисунок А.7 – Шляхи розв'язання проблеми (2)

Узагальнена модель SOA додатку

1. Додаток може використовувати функції λ , що надають сервіси F SaaS-рівня:

$$F = \{f_j\}, j = \overline{1, k}.$$

2. Виконання цих функцій забезпечують сервіси P PaaS-рівня:

$$P = \{p_{ij}\}, i = \overline{1, k}; j = \overline{1, m}.$$

3. Сервіси A IaaS-рівня забезпечують апаратну платформу:

$$A = \{a_{ijs}\}, \\ i = \overline{1, k}; j = \overline{1, m}; s = \overline{1, n}.$$

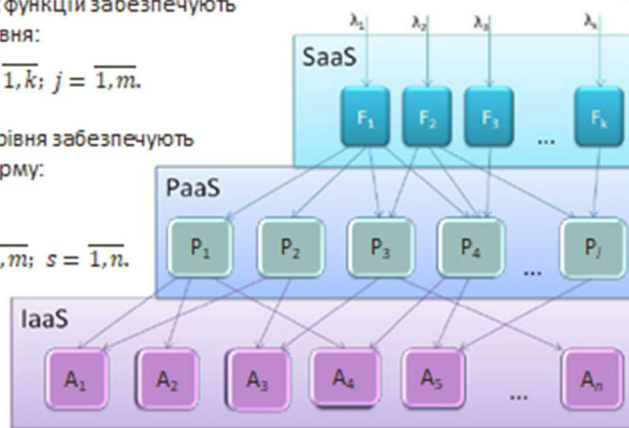


Рисунок А.8 – Узагальнена модель додатку (1)

Узагальнена модель SOA додатку

- Множина функцій додатку, що можуть надавати різні платформи та сервіси:

$$\hat{F} = \lambda \cup FAP$$

- У приватній хмарі формуємо віртуальний набір функцій $\hat{V}R$, який буде забезпечувати надання сервісів хмарному застосуванню:

$$E: U \times \Psi \times R \times \hat{V}R \times O \times \Theta,$$

де

U – множина вхідних даних предметної області;

O – множина вихідних даних;

Ψ – конфігурація запуску додатку;

Θ – множина службових вихідних даних;

E – функція виконання ПЗ, яке використовує хмарні сервіси.

Рисунок А.9 – Узагальнена модель додатку (2)

Модель розгортання

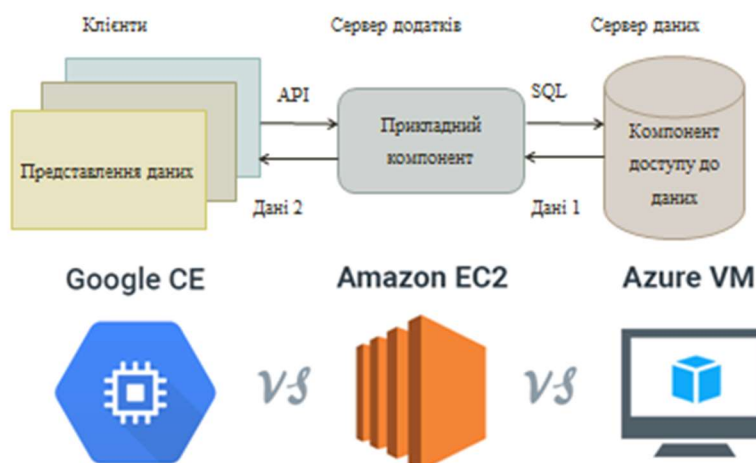


Рисунок А.10 – Модель розгортання

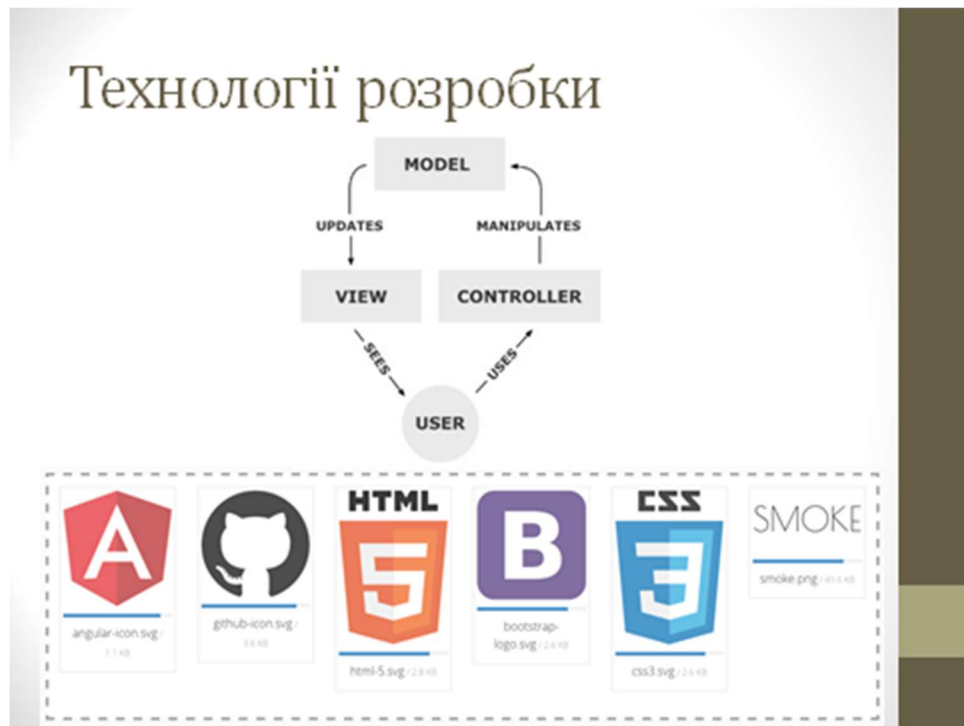


Рисунок А.11 – Технології розробки



Рисунок А.12 – Архітектура системи



Рисунок А.13 – Технологічні рішення

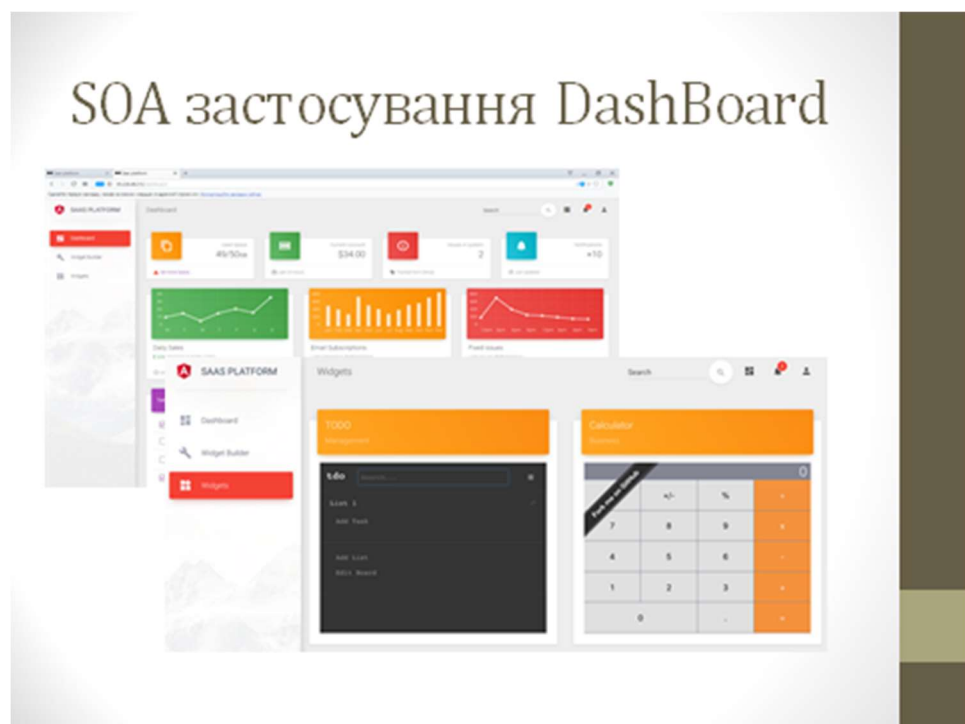


Рисунок А.14 – SOA застосування



Рисунок А.15 – Висновки по роботі

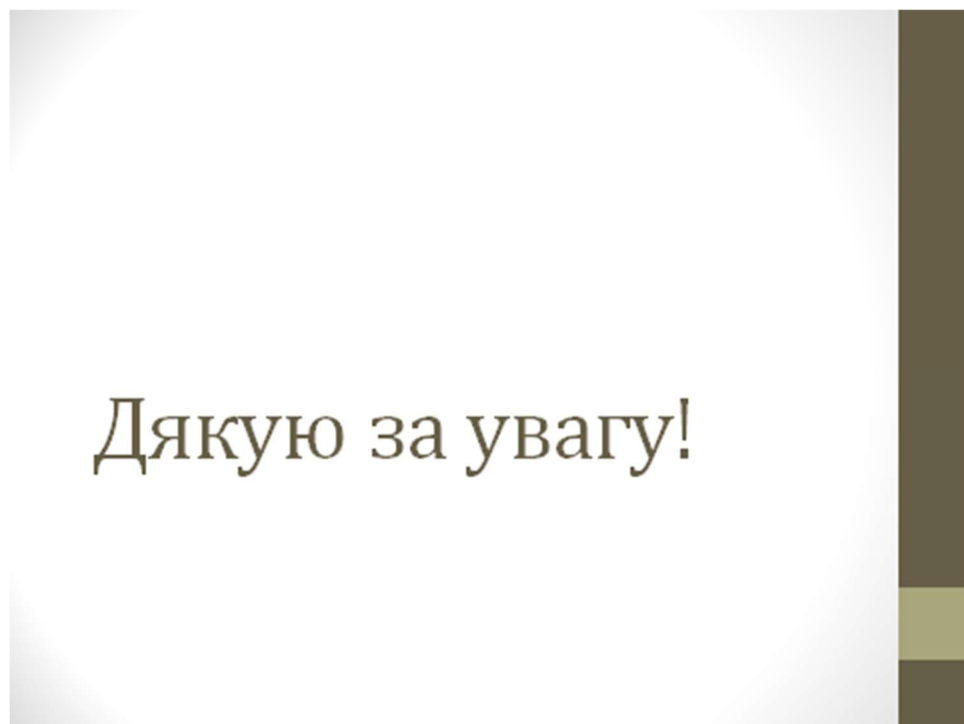


Рисунок А.16 – Кінцевий слайд

ДОДАТОК Б

Програмні коди

Б.1 Сутність, що відповідає за модель вакансії

```
@Entity
@Data
public class Job {

    @Id
    private String url;
    private String title;
    // Max value for PostgreSQL
    @Column(length = 10485760)
    private String description;
    private String company;
    private String source;
    private LocalDateTime date;
    private String dateToDisplay;

    public Job() {
    }

    public Job(String title, String description, String company, String
source, String url, LocalDateTime date) {
        this.title = title;
        this.description = description;
        this.company = company;
        this.source = source;
        this.url = url;
        this.date = date;
    }

    public String getDateToDisplay() {
        return date.format(ofPattern("d MMMM"));
    }
}
```

Б.2 Базовий клас для парсингу інформації

```
public class JobParser {

public static final String NBSP = "\u00a0";

JobSite jobSite;

public JobParser(JobSite jobSite) {
this.jobSite = jobSite;
}

public Document getDoc(String siteUrl) throws ParserException {
try {
return
Jsoup.connect(siteUrl).userAgent("Mozilla").timeout(0).get();
} catch (IOException e) {
throw new ParserException("Failed connecting to " + siteUrl + "\n"
+ e.getMessage());
}
}

public String getUrl(Elements titleBlock) {
return jobSite.urlPrefix() + titleBlock.attr("href");
}

public Elements getJobBlocks(Document doc) throws ParserException {
Elements jobBlocks = getElements(doc, jobSite.jobBox());
check(jobBlocks, "job blocks");
return jobBlocks;
}

public Elements getTitleBlock(Element job) throws ParserException {
Elements titleBlock = getElements(job, jobSite.titleBox());
check(titleBlock, "title blocks");
return titleBlock;
}

public String getTitle(Elements titleBlock) {
return titleBlock.text();
}

public String getDescription(Element job, String url) throws
ParserException {
return getElements(job, jobSite.description()).text();
}

public String getCompany(Element job, String url) throws
ParserException {
```

```

    String company = removeNbsp(getElements(job,
jobSite.company()).text());
    check(company, "company", url);
    return company;
}

    public LocalDateTime getDate(Element job, String url) throws
ParseException {
    String dateLine = getElements(job, jobSite.date()).text();
    check(dateLine, "date", url);
    return getDateByLine(dateLine);
}

    protected LocalDateTime getDateByLine(String dateLine) {
    String[] dateParts = dateLine.split(jobSite.split());
    MonthsTools.removeZero(dateParts);
    return LocalDate.of(parseInt(dateParts[2]), parseInt(dateParts[1]),
parseInt(dateParts[0])).atTime(getTime());
}

    protected LocalTime getTime() {
    return LocalTime.now(ZoneId.of("Europe/Athens"));
}

    //in case we parse in January jobs of last December. Needed for
jobs.ua and hh.ua
    int getYear(int month) {
    if (month >
LocalDate.now(ZoneId.of("Europe/Athens")).getMonthValue())
    return LocalDate.now().getYear() - 1;
    return LocalDate.now(ZoneId.of("Europe/Athens")).getYear();
}

    Elements getElements(Element element, JobSite.Holder holder) {
    return getElements(element, holder, false);
}

    Elements getElements(Element element, JobSite.Holder holder,
boolean starting) {
    if (starting)
    return element.getElementsByAttributeValueStarting(holder.key,
holder.value);
    return element.getElementsByAttributeValue(holder.key,
holder.value);
}

    String removeNbsp(String text) {
    return text.replaceAll(NBSP, "");
}

    void check(Object o, String data) throws ParseException {
    check(o, data, null);
}

```

```

void check(Object o, String data, String url) throws
ParseException {
    String jobUrl = url == null ? "" : url;
    if (o == null || o.toString().trim().length() == 0) {
        log.error("Error getting {} from {}, {}", data, jobSite.name(),
jobUrl);
        throw new ParseException("Error getting " + data + " from " +
jobSite.name() + "\n" + jobUrl);
    }
}

private static final Logger log =
LoggerFactory.getLogger(Parser.class);
}

```

Б.3 Сервіс, що зберігає та оновлює вакансії

```

@Service
public class JobServiceImpl implements JobService {

    private JobRepository jobRepository;
    private JTwitter twitter;
    private Notifier notifier;

    @Autowired
    public JobServiceImpl(JobRepository jobRepository, JTwitter
twitter, Notifier notifier) {
        this.jobRepository = jobRepository;
        this.twitter = twitter;
        this.notifier = notifier;
    }

    public void save(Job job) {
        if (jobRepository.exists(job.getUrl())) {
            update(job);
        } else {
            saveJob(job);
            log.info("New job '{}' on {} found", job.getTitle(),
job.getSource());
        }
    }

    private void update(Job job) {
        Job jobFromDb = jobRepository.findOne(job.getUrl());
        LocalDate jobFromDbDate = jobFromDb.getDate().toLocalDate();
        LocalDate jobDate = job.getDate().toLocalDate();
    }
}

```

```

if (!jobFromDbDate.equals(jobDate)) {
    saveJob(job);
    log.info("Job '{}', {}, was updated", job.getTitle(),
job.getUrl());
}
}

public Page<Job> getJobs(Pageable request) {
return jobRepository.findAll(request);
}

private void saveJob(Job job) {
try {
jobRepository.save(job);
} catch (Exception e) {
log.error("Error while saving job '{}', {} into database",
job.getTitle(), job.getUrl(), e);
// notifier.notifyAdmin("Error while saving following job into
database: '" +
// job.getTitle() + "', " + job.getUrl() + "\n\n" + e.getMessage());
}
}

private static final Logger log =
LoggerFactory.getLogger(JobServiceImpl.class);
}

```

Б.4 Клас, що відповідає за формування основної сторінки веб сервісу

```

@Controller
public class ParseController {

private static final int PAGE_SIZE = 40;
private JobService jobService;

@Autowired
public ParseController(JobService jobService) {
this.jobService = jobService;
}

@RequestMapping(value = "/", method = RequestMethod.GET)
public ModelAndView showJobs(@RequestParam(value = "page", required
= false) Integer page) {

ModelAndView modelAndView = new ModelAndView("index");
int currentPageNumber = (page == null || page < 1) ? 0 : page - 1;

```

```
Pageable request = new PageRequest(currentPageNumber, PAGE_SIZE,
Sort.Direction.DESC, "date");
Page<Job> jobs = jobService.getJobs(request);
PageBox pageBox = new PageBox(jobs.getTotalPages(),
jobs.getNumber());

modelAndView.addObject("jobs", jobs);
modelAndView.addObject("pageBox", pageBox.getPageBox());

return modelAndView;
}

@RequestMapping("/about")
public String about() {
return "about";
}
}
```