Розробка серверної частини мобільного додатку організації служби донорства крові з використанням мікросервісної архітектури

Автор роботи:

Студент гр. ДА-61

Грудович Володимир Іванович

Науковий керівник:

Асистент кафедри СП

Науменко Тетяна Олександрівна

Об'єкт дослідження

Мікросервісна архітектура; ситуація з організацією служби донорства крові.

Предмет дослідження

Шаблони створення мікросервісної архітектури, засоби взаємодії мікросервісів. Розробка додатку донорства крові.

Мета та завдання

Метою даної роботи є:

- дослідження шаблонів створення мікросервісної архітектури, аналіз існуючих засобів для реалізації мікросервісів
- перевірка на практиці позитивних та негативних аспектів використання архітектури мікросервісів

Завданням даної роботи є:

- реалізація серверної частини додатку, що буде спрощувати організацію донорства крові з використанням шаблонів побудови архітектури мікросервісів.

Актуальність роботи

- В Україні кількість донацій крові майже втричі менша (12 на 1000 населення), ніж рекомендовано Всесвітньою організацією охорони здоров'я (33 на 1000 населення).
- через недосконалість системи служби крові 40% заготовленого матеріалу знищується.
- відсутність дистанційної взаємодій донорів та слубжи здачі крові призводить до незацікавленості донорів здійснювати повторну здачу.
- Мікросервісна архітектура відкриває широкі можливості для масшабування та розширення функціоналу додтка у майбутньому.

Етапи виконання

- Дослідження ситуації у сфері донорства крові
- Аналіз існуючих проблем
- Розробка рішень, для вирішення ключових проблем
- Аналіз концепції мікросервісної архітектури в цілому
- Аналіз шаблонів реалізації мікросервісної архітектури
- Розробка архітекутри додатку
- Висновки з отриманої роботи

Матеріали та засоби розробки, що використовувались

- Медичні данні, що знаходяться у відкритому доступі
- 17 робіт, що стосуються розробки архітектури додатку та реалізації MSA
- Шаблони реалізаці MSA
- Програмні засоби Java та допоміжні бібліотеки

Організаційна частина

Основні цілі реалізації додатку:

- Покращити якість медичного обслуговування донорів та реципієнтів;
- Зробити процес пошуку донорів та реципієнтів швидким;
- Маючи необхідну доступну базу даних донорів та реципієнтів зробити процес здачі та отримання крові швидким;
- Зменшити навантаження на лікарів ввівши попередній електонний запис на процедуру здачі крові.

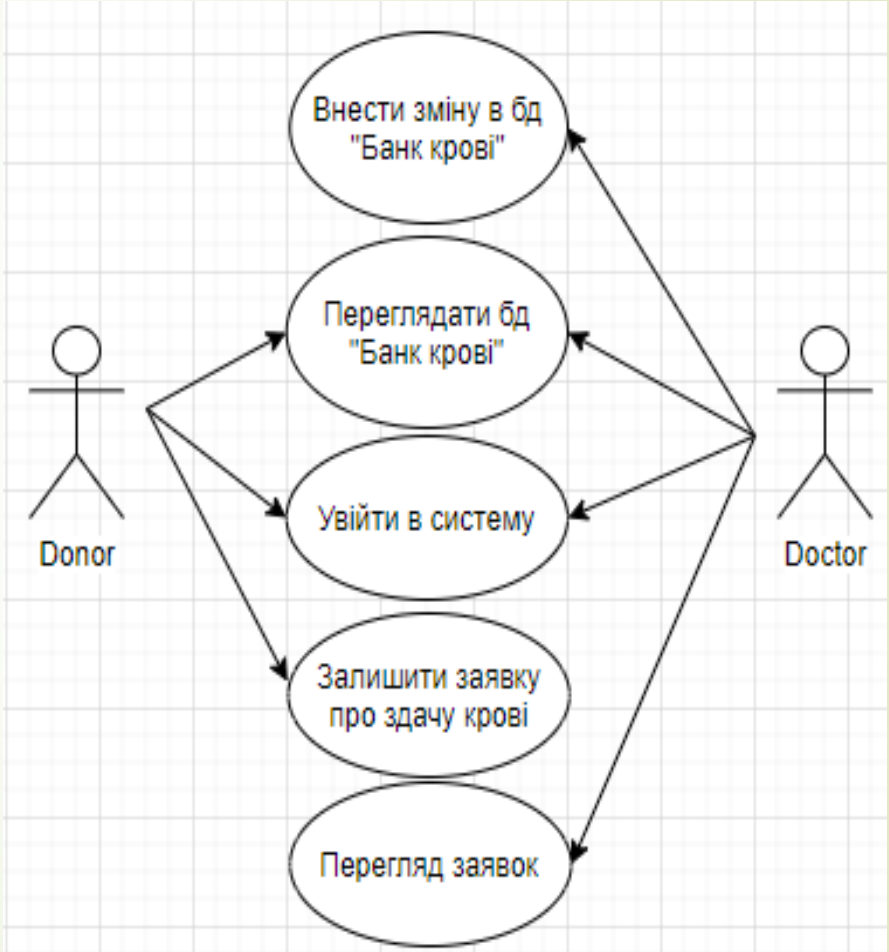
Зацікавлені сторони:

- Медичні заклади приватного та муніципального типу;
- Донори та реципієнти крові
- Люди усіх соціальних груп

Організаційна частина

Діаграма прецедентів

- Основною ідеєю додатку є спрощення організації донорства крові, тому в додатку має бути мінімум 2 дійові особи: лікар та донор.
- Лікар відповдальний за «Банк крові», вносить зміни, корегує вже існуючі дані.
- Донор може переглядати БД «Банк крові» і оформляти заявки, про намір здати кров, які в свою чергу зможе переглядати лікар.

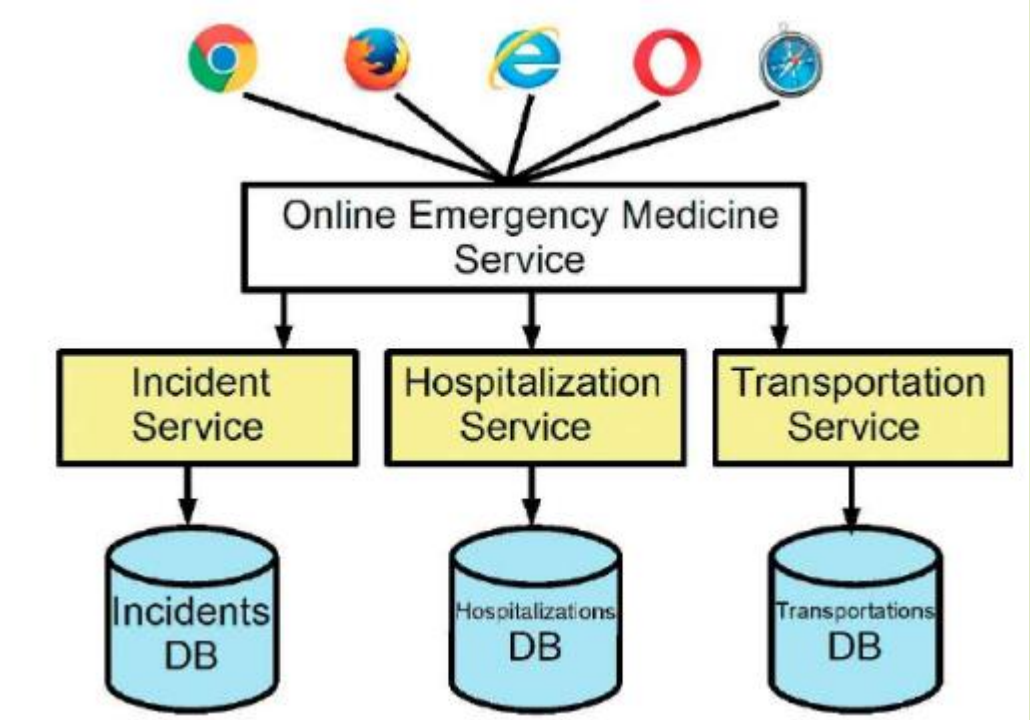


Коротко про мікросервісну архітектуру

Мікросервісна архітектура (MicroServices Architecture, MSA) – архітектурний стиль програмної розробки, за яким система будується як набір слабкозвязаних модулів – сервісів, що виконують в окремому процесі одну поставлену перед ними бізнес задачу

Основні властивості MSA

- низька складність у розробці
- сфокусованість (Single Responsibility Principle)
- слабке зв'язування (Dependency Injection)
- сильне зчеплення



Засоби реалізації

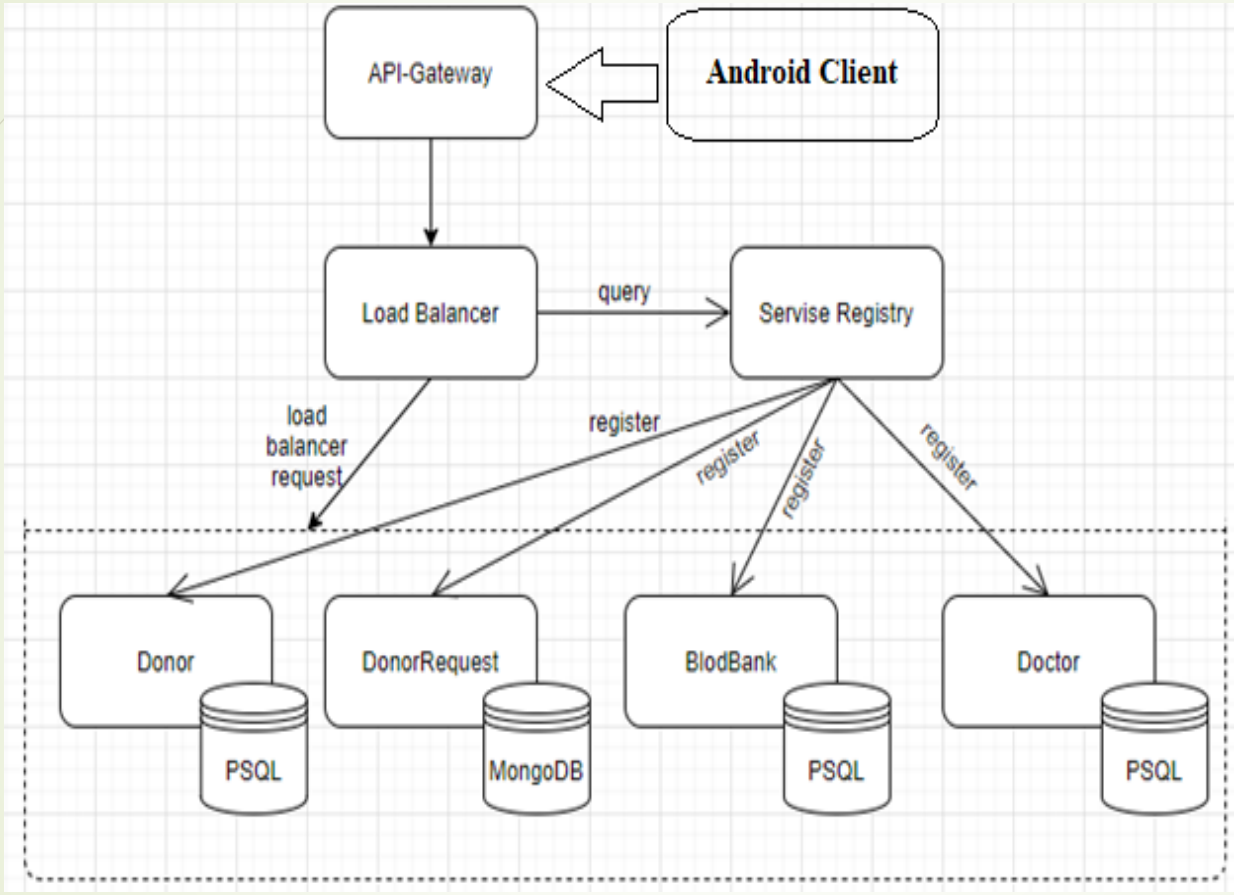
Інструментні

- Фреймворк Spring
- Бази даних :
  - Реляційна – PostgreSQL
  - Нереляційна – MongoDB
- Засоби створення мікросервісної взаємодії: Netflix: Zuul, Eureka, Ribbon
- Середовище розгортання Docker
- Android Client

Шаблонні

- Service Registry (реєстр сервісів) - Eureka
- Load Balancer (балансувальник навантаження) - Ribbon
- API Gateway (єдина точка входу/шлюз) - Zuul

Розробка архітектури додатку



Службові сервіси:

- API-Gateway
- Load Balancer
- Servise Registry

Сервіси, що реалізують бізнес-логіку:

- Donor
- DonorRequest
- BlodBank
- Doctor

Вся взаємодія між сервісами є синхронною та відбувається шляхом звертання до певної точки REST API кожного сервісу

API-Gateway

- **API Gateay** – це сервіс, що реалізує точку входу для клієнта й дає доступ до використання данних, бізнес-логіки та функціональних можливостей сервісів.
- **Netflix Zuul** – це зручна та ефективна реалізація APIG від компанії Netflix
- Для запуску Zuul потрібно вказати лише анотацію **@EnableZuulProxy**.

```
zuul:
    ignoredServices: "*"
    routes:
    doctor: /doctor/
    donor: /donor/
    blodBank: /blodBank/
```

```
@SpringBootApplication
@Controller
@EnableZuulProxy
public class ZuulServerApplication {
    public static void main(String[] args) {
        new SpringApplicationBuilder(ZuulServerApplication.class)
            .web(true).run(args);
    }
}
```

Servise Registry

- **Netflix Eureka** – це сервіс, що реалізує реєстрацію та збереження статусу всіх сервісів у системі
- Може здійснювати керування окремими сервісами сервісів

```
$ open $(echo \"(echo $DOCKER_HOST)/donor\")
\sed 's/tcp:\/\//http:\/\//g'|
\sed 's/[0-9]\{4,\}/10000/g'|
\sed 's/\"//g')
```

```
"_self": {
"href": "http://127.0.0.1:1000/donor"
},
"resume":{
"href": "http://127.0.0.1:1000/donor/resume"
},
```

Load Balancer

Netflix Ribbon – балансувальник навантаження розроблений компанією Netflix. Він автоматично взаємодіє з Netflix Eureka, що полегшує розробку нашої системи

Причини використання:

- враховуючи, що наша система буде розширюватись у майбутньому, нам життєво необхідний такий механізм, що буде слідкувати за відмовостійкістю, а також кешувати данні

```
donor-service:
  ribbon:
    eureka:
      enabled: true
    ServerListRefreshInterval: 1000
donor-service:
  ribbon:
    listOfServers: localhost:8085,localhost:8086
    eureka:
      enabled: true
```

```
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
@RibbonClient(name = "donor-service", configuration = RibbonConfiguration.class)
public class DonorServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(DonorServiceApplication.class, args);
    }
    @Bean
    @LoadBalanced
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
    public @Bean
    WebClient webClient() {
        return WebClient.builder().clientConnector(new ReactorClientHttpConnector())
        .baseUrl("http://localhost:8081").build();
    }
}
```

Donor service

- Donor service необхідний нам для реалізації взаємодії з донором, він відповідальний за реєстрацію, аутентифікацію та логування сутності донора.
- Донор взаємодіє з сервісом DonorRequest, а також може зчитувати інформацію з сервісу BoldBank.
- У якості збереження донорів використовує БД PostgreSQL

```java
@RequestMapping("/donors/")
public class UserRestController {
    @Autowired
    UserService donorService;

    @RequestMapping(value = "{id}", method = RequestMethod.GET,
                    produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<donor> getUser(@PathVariable("id")Long id){
        if(id == null){
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
        Donor donor = this.donorService.getById(id);

        if (donor == null){
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
        return new ResponseEntity<>(donor, HttpStatus.OK);
    }
}
```

```java
public class CustomAuthentication implements AuthenticationProvider {
    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationException {
      String username = authentication.getName();
      String password = authentication.getCredentials().toString();
      if ("user".equals(username) && "password".equals(password)) {
          return new UsernamePasswordAuthenticationToken(username, password, Collections.emptyList());
       } else {
          throw new BadCredentialsException("Authentication failed");
       }
    }
    @Override
    public boolean supports(Class<?>aClass) {
        return aClass.equals(UsernamePasswordAuthenticationToken.class);
    }
}
```

DonorRequest service

- Даний сервіс реалізує подання заявки на здачу крові донором. Він відповідає за створення / читання / зміну / видалення (CRUD) заявок . Взаємодія з сервісом Donor та Doctor відбувається через REST інтерфейс по протоколу HTTP.
- В якості БД збереження даних використовуэться NoSQL база даних MongoDB

```
@RepositoryRestResource(collectionResourceRel = “donorRequest”, path = “donorRequest”)
public interface GenreRepository extends
PagingAndSortingRepository<DonorRequest, Long> {
List<DonorRequest> findByName(@Param(“0”) String name);
}
```

```
$ curl -i -X POST -H “Content-Type:application/json”
-d “{\”type\”:\”thirdGroup\”}” http://localhost/donorRequest
```

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Location: http://localhost/DonorRequest/1
Content-Length: 0
```

BlodBank service

- Даний сервіс відповідає за створення / читання / зміну / видалення (CRUD) інформації в банку крові.
- Данні банку крові вказують на те який запас крові і якої групи є в належності.
- У якості збереження інформації про наявні види крові використовує БД PostgreSQL.
- Взаємодія з данними відбувається завдяки Spring Data й використовується шаблон репозиторію, який взаємодіє з сутностями (Entity).
- Інтерфейс користувача наслідується від інтерфейсу JpaRepository

```
public class UserServiceImpl implements UserService {
    @Autowired
    UsersRepository BlodBankRepository ;
    @Override
    public User getById(long id) {return BlodBankRepository .getOne(id);}
    @Override
    public void save(Blod user) {BlodBankRepository .save(user);}
    @Override
    public void delete(long id) {BlodBankRepository .deleteById(id);}
    @Override
    public List<Blod> getAllUsers() {return BlodBankRepository .findAll();}
}
```

Doctor service

- Doctor service необхідний нам для реалізації взаємодії з лікарем.
- Даний сервіс проводить реєстрацію, аутентифікацію та логування.
- Doctor взаємодіє з сервісом BlodBank, а також може зчитувати інформацію з сервісу DonorRequest за допомогою REST інтерфейсу по протоколу HTTP

```
@RequestMapping(value = "", method = RequestMethod.POST, produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity< blodType > saveBlodType(@RequestBody @Valid BlodType blodBank){
    HttpHeaders headers = new HttpHeaders();
    if(blodType == null){
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
    this.blodTypeService.save(blodType);
    return new ResponseEntity<>( blodType, headers, HttpStatus.CREATED);
}
```

```
@RequestMapping(value = "", method = RequestMethod.PUT, produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity< blodType > updateBlodType(@RequestBody @Valid BlodType blodType){
    HttpHeaders headers = new HttpHeaders();
    if(doctor == null){
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
    this.blodTypeService.save(blodType);
    return new ResponseEntity<>( blodType, headers, HttpStatus.OK);
}
```

Docker

- **Docker** – це платформа для розробки, зборки, а також введення в експлуатацію додатків. За допомогою докера можливо відділити систему від інфраструктури системи й керувати інфраструктурою, як окремим додатком.

Переваги докера:

- Швидкий запуск
- Швидке розгортання
- Зручне управління та масштабування
- Працює з більшістю систем
- Грамотний розподіл ресурсів

Недоліки докера:

- Для грамотного використання потрібний DevOps
- Докер для роботи контейнерів потребує ресурси

Висновки

Переваги мікросервісного та недоліки мікросервісного підходу до розробки ІС

Переваги

- Технологчіна різнорідність
- Стійкість
- Масштабованість
- Незалежне розгортання
- Компонованість
- Організація роботи

Недоліки

- Складність в підборі та проектуванні сервісів
- Складність розробки
- Проблеми взаємодії
- Керування данними
- Ресурсонеобхідність
- Тестування й моніторинг

ДЯКУЮ ЗА УВАГУ