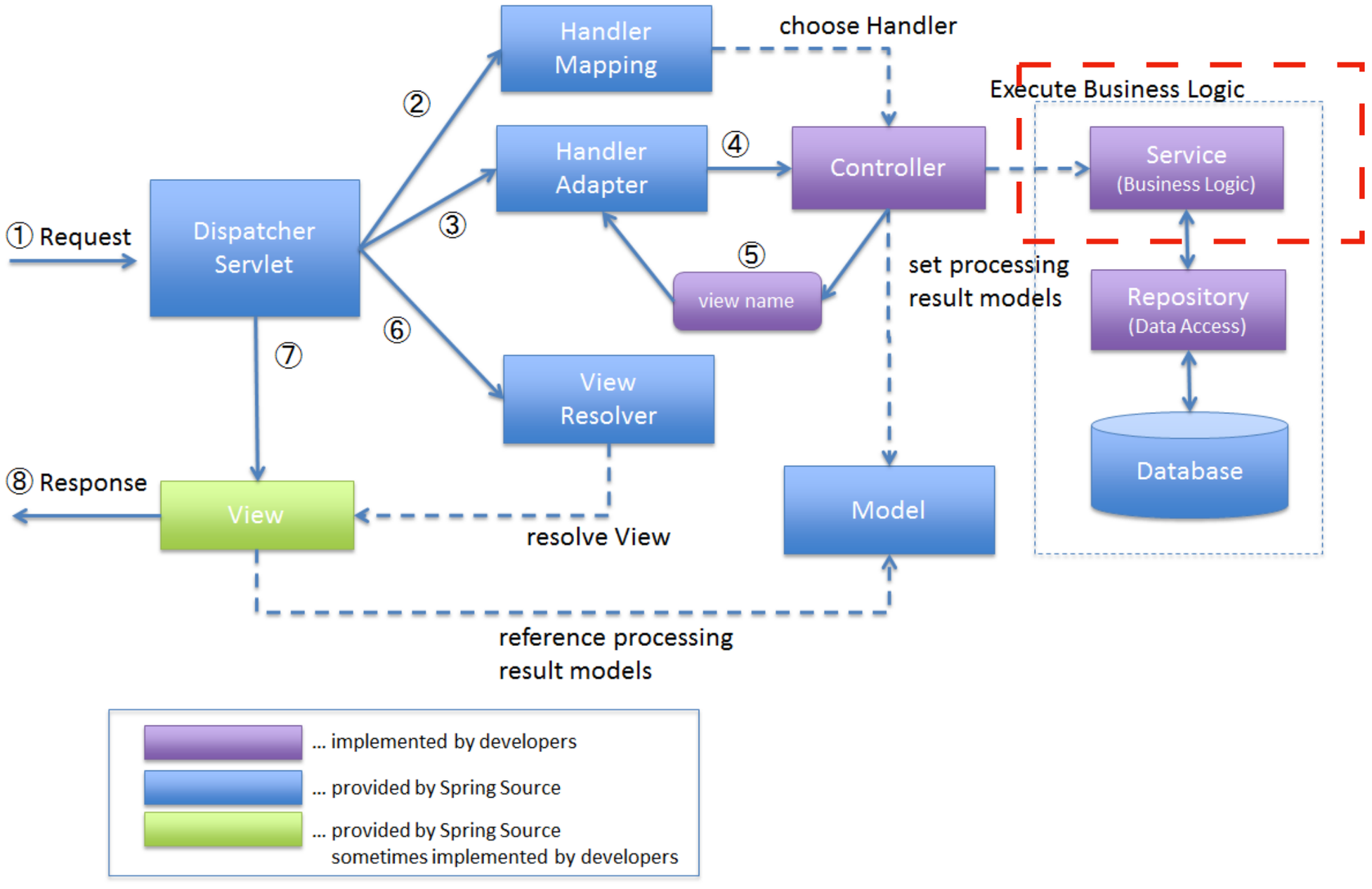
Spring Web MVC

Servisni sloj

Spring Web MVC
tok obrade zahteva - detaljnije



Slika preuzeta sa: https://terasolunaorg.github.io/guideline/5.0.x/en/Overview/SpringMVCOverview.html

Spring Web MVC
Servisni sloj

- Servisni sloj se koristi kao omotač (wrapper) oko sloja za pristup podacima (DAO) i predstavlja opšteprihvaćen način za korišćenje DAO sloja
- Može biti izostavljen tako što ćemo koristiti DAO sloj direktno, ali njegovo uvođenje može doneti mnoge pogodnosti:
  - U servisni sloj se stavljaju sve CRUD (create, retrieve, update, delete) metode koje repository nudi.
    Zašto ne koristiti direktno *Repository*? *Repository* nudi samo osnovne operacije. Ako želimo da proverimo polovnu logiku u tim podacima to nije moguće. Takve provere možemo vršiti u metodama u servisnom sloju
- Preporuka je da se u Service sloju piše **poslovna logika** sistema - manipulacija domenskim podacima, tako da se kontroleri samo obraćaju servisnom sloju da odradi određene poslovne operacije, a servisi po potrebi pristupaju DAO sloju. Na ovaj način se kontroleri pojednostavljuju, alogika aplikacije smešta u servisni sloj.

Spring Web MVC
Servisni sloj

```
@Service
public class StudentService {

    @Autowired
    StudentRepository studentRepository;

    public Student findOne(Long id) {
        return studentRepository.findOne(id);
    }

    public List<Student> findAll() {
        return studentRepository.findAll();
    }
}
```

Anotacija označava da ova komponenta pripada servisnom sloju

Ovaj objekat nikada nije instanciran sa StudentRepository studentRepository = new StudentRepository();

Kako možemo koristiti objekat za koji nikada nije zauzeta memorija i jednak je „null“? Zar nećemo izazvati Exception?

@Autowired anotacija rešava problem i ovaj mehanizam je poznat pod nazivom **DependencyInjection**.

Spring Web MVC
Servis - deklaracija interfejsa

- Uobičajeno je da se deklariše interfejs sa željenim metodama a zatim on implementira

```
public interface ProductService {
  public abstract void createProduct(Product product);
  public abstract void updateProduct(String id, Product product);
  public abstract void deleteProduct(String id);
  public abstract Collection<Product> getProducts();
}
```

Spring Web MVC
Servis - implementacija

```
@Service
public class ProductServiceImpl implements ProductService {
  @Autowired
  ProductRepository productRepo;
  ...
  @Override
  public void createProduct(Product product) {
    productRepo.put(product.getId(), product);
  }
  @Override
  public void updateProduct(String id, Product product) {
     productRepo.remove(id);
     product.setId(id);
     productRepo.put(id, product);
  }
  @Override
  public void deleteProduct(String id) {
     productRepo.remove(id);

  }
  @Override
  public Collection<Product> getProducts() {
    return  productRepo.values();
  }
}
```

Spring Web MVC
Servisni sloj - primer pristupa iz kontrolera

```
@RestController
public class ProductServiceController {
  @Autowired
  ProductService productService;

  @RequestMapping(value = "/products")
  public ResponseEntity<Object> getProduct() {
    return new ResponseEntity<>(productService.getProducts(), HttpStatus.OK);
  }
  @RequestMapping(value = "/products/{id}", method = RequestMethod.PUT)
  public ResponseEntity<Object>
    updateProduct(@PathVariable("id") String id, @RequestBody Product product) {

    productService.updateProduct(id, product);
    return new ResponseEntity<>("Product is updated successsfully", HttpStatus.OK);
  }
  @RequestMapping(value = "/products/{id}", method = RequestMethod.DELETE)
  public ResponseEntity<Object> delete(@PathVariable("id") String id) {
    productService.deleteProduct(id);
    return new ResponseEntity<>("Product is deleted successsfully", HttpStatus.OK);
  }
  @RequestMapping(value = "/products", method = RequestMethod.POST)
  public ResponseEntity<Object> createProduct(@RequestBody Product product) {
    productService.createProduct(product);
    return new ResponseEntity<>("Product is created successfully", HttpStatus.CREATED);
  }
}
```