



Università
di Catania



A Microservice Application

Alessandro Midolo, PhD

Dipartimento di Matematica e Informatica

Università di Catania

alessandro.midolo@unict.it

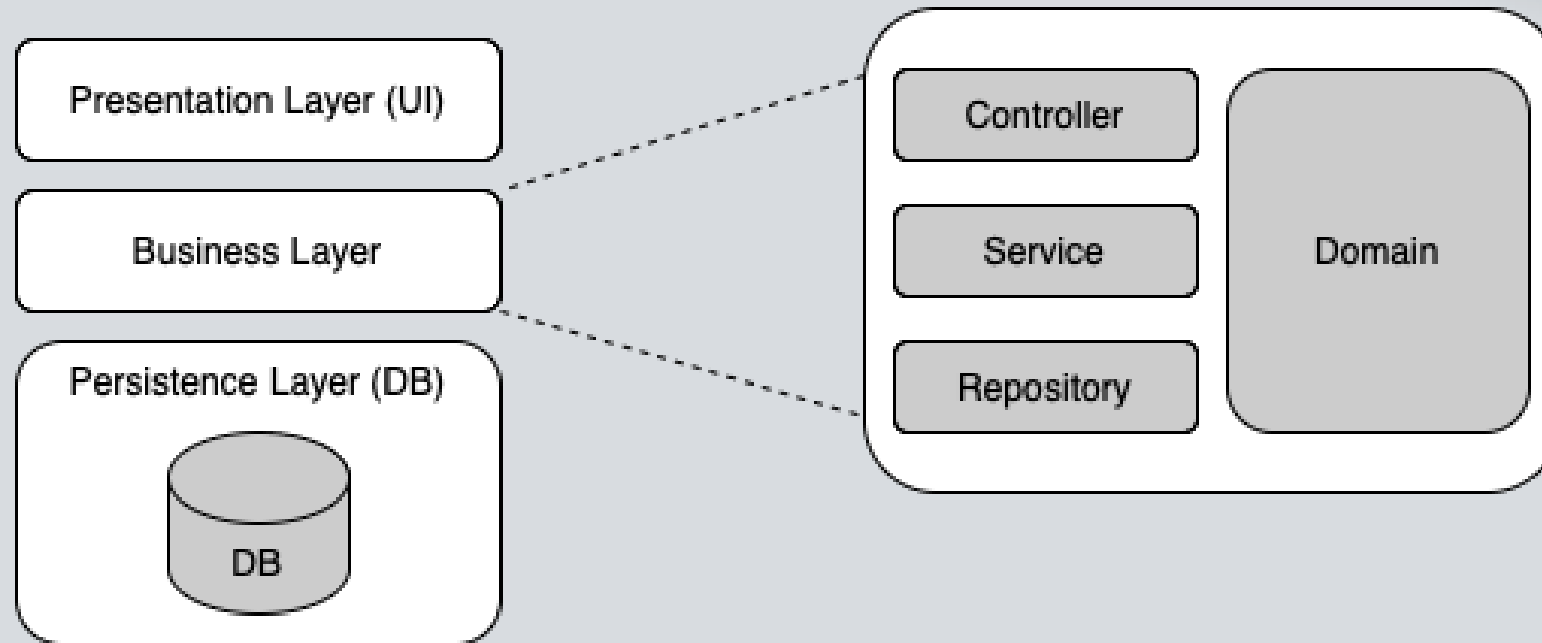
<https://www.dmi.unict.it/amidolo/>

A.A. 2024/2025

Use Case: Space Scanner

- Vogliamo creare un servizio chiamato Space Scanner che permette di **scoprire nuove forme di vita aliena** sui pianeti della galassia!
- Inizieremo con due pianeti esplorabili: **Marte e Venere**
- Ogni **alieno** è descritto da un **nome**, una **razza** e un **pianeta** di “residenza”
- Atterrando su un pianeta potremo scoprire solo alieni che vivono **su quel pianeta**
- La razza non deve necessariamente coincidere con la residenza

Architettura a Microservizi



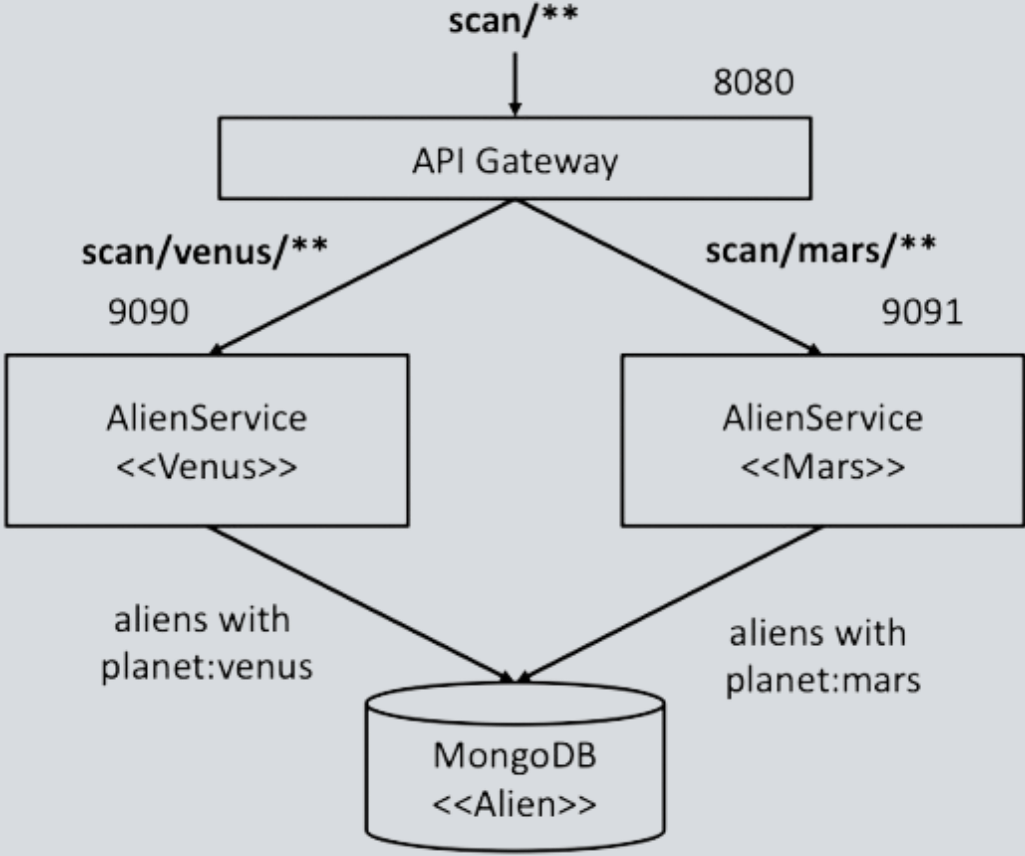
Controller: interfaccia con il client (UI), espone le API (REST)

Service: implementa la logica di business

Repository: interfaccia con il DB, espone metodi per l'interrogazione in logica OOP

Domain: contiene le entità (oggetti) per la rappresentazione e lo scambio dei dati

Panoramica Microservizi



Outline

Repository: <https://github.com/Laboratorio-ISD/FinalMicroservice>

- **step 1:** Unzip alien.zip
- **step 2:** Hello World!
- **step 3:** Hello Planet! (Autowire)
- **step 4:** Properties
- **step 5:** MongoDB con Docker
- **step 6:** Some REST APIs
- **step 7:** Alien CRUD REST APIs
- **step 8:** Zuul API Gateway
- **step 9:** Dockerize

1. Inizializziamo il Progetto

- Navigate all'url <https://start.spring.io/> per inizializzare il progetto
- Scegliere Maven e java come linguaggio
- Selezionare le dipendenze **Spring Web**
- Generare il progetto
- Scaricare il file zip ed estrarlo all'interno del proprio IDE

EXPLORER

- ALIENT
 - .mvn
 - .vscode
 - src
 - main
 - java/com/labisd/alien
 - controller
 - HelloController.java**
 - AlienApplication.java
 - resources
 - test
 - target
 - .gitattributes
 - .gitignore
 - HELP.md
 - mvnw
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN

AlienApplication.java HelloController.java

```
src > main > java > com > labisd > alien > controller > HelloController.java > ...
1 package com.labisd.alien.controller;
2
3 import org.springframework.web.bind.annotation.RestController;
4 import org.springframework.web.bind.annotation.RequestMapping;
5
6 @RestController
7 public class HelloController {
8     @RequestMapping("/")
9     public String index() {
10         return "Greetings from World!";
11     }
12 }
13
14
```

2. Hello World
Definiamo il nostro controller base che gestisce tutte le richieste HTTP sul percorso `http://localhost:8080` attraverso il metodo `index()`

3. Hello Planet (@Autowired)

- Vogliamo fare in modo che il messaggio di benvenuto sia personalizzato in base al “pianeta” di atterraggio (servizio Alien<<Mars>> o Alien<<Venus>>)
- Definiamo un’interfaccia **Planet** e due classi **Mars** e **Venus** che la implementano
- Quale delle due istanza usare verrà decisa a **runtime** tramite **Dependency Injection**
- Con **@Autowired** viene chiesto al framework di inserire la dipendenza corretta per
- Planet con il **bean** opportuno (a runtime)

```
package com.labisd.alien.bean;
```

```
public interface Planet {  
    public String getGreetings();  
    public String getName();  
}
```

Creiamo quindi l'interfaccia **Planet** e le due classi **Mars** e **Venus** che implementano due versioni differenti dell'interfaccia

```
package com.labisd.alien.bean;
```

```
public class Mars implements Planet {  
  
    @Override  
    public String getGreetings() {  
        return "Red Planet";  
    }  
  
    @Override  
    public String getName() {  
        return "mars";  
    }  
}
```

```
package com.labisd.alien.bean;
```

```
public class Venus implements Planet {  
  
    @Override  
    public String getGreetings() {  
        return "Yellow Planet";  
    }  
  
    @Override  
    public String getName() {  
        return "venus";  
    }  
}
```

EXPLORER

- ALIENT
- > .mvn
- > .vscode
- src
 - main
 - java/com/labisd/alien
 - bean
 - Mars.java
 - Planet.java
 - Venus.java
 - controller
 - HelloController.java
 - AlienApplication.java
 - resources
 - test
 - target
 - .gitattributes
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
- OUTLINE
- TIMELINE
- JAVA PROJECTS
- MAVEN

Tab bar showing: AlienApplication.java, HelloController.java (active), Planet.java, Mars.java, Venus.java

```

src > main > java > com > labisd > alien > controller > HelloController.java > ...
1  package com.labisd.alien.controller;
2
3  import org.springframework.web.bind.annotation.RestController;
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.web.bind.annotation.RequestMapping;
6
7  import com.labisd.alien.bean.Planet;
8
9  @RestController
10 public class HelloController {
11
12     @Autowired
13     Planet planet;
14
15     @RequestMapping("/")
16     public String index() {
17         return "Greetings from " + planet.getGreetings() + "!";
18     }
19 }
20

```

- Vogliamo fare in modo che il messaggio di benvenuto sia personalizzato in base al "piante" di atterraggio (servizio Alien<<Mars>> o Alien<<Venus>>)
- Usiamo l'interfaccia **Planet** e le due classi **Mars** e **Venus** che la implementano
- Quale delle due istanze usare verrà deciso a **runtime** tramite **Dependency Injection**
- Con **@Autowire** viene chiesto al framework di inserire la dipendenza corretta per Planet con il **bean** opportuno (a runtime)

EXPLORER

- ALLEN
- > .mvn
- > .vscode
- src
 - main
 - java/com/labisd/alien
 - bean
 - Mars.java
 - Planet.java
 - Venus.java
 - controller
 - HelloController.java
 - AlienApplication.java
 - resources
 - test
 - target
 - .gitattributes
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN

AlienApplication.java × HelloController.java Planet.java Mars.java Venus.java

```
src > main > java > com > labisd > alien > AlienApplication.java > ...
7   import com.labisd.alien.bean.*;
8
9   @SpringBootApplication
10  public class AlienApplication {
11
12      private String planet = "venus";
13
14      public static void main(String[] args) {
15          SpringApplication.run(AlienApplication.class, args);
16      }
17
18      // @Bean vanno definiti in una classe @Configuration(@SpringBootApplication)
19      @Bean
20      public Planet planet(){
21          System.out.println("Landing planet: " + planet);
22
23          switch (planet) {
24              case "mars":
25                  return new Mars();
26              case "venus":
27                  return new Venus();
28              default:
29                  return new Mars();
30          }
31      }
32  }
33
```

- Il metodo **@Bean** di una classe **@Configuration** viene chiamato dal framework all'avvio
- L'istanza restituita viene inserita nello **Spring Container**
- A runtime verrà definito quale classe instanziare (**Mars** o **Venus**)

4. @Value e le Properties

- Attualmente la scelta di Planet è hardwired nel codice
 - `http://localhost:8080` restituisce sempre ***Greetings from Yellow Planet!!***
- Possiamo usare le **properties** per configurare il servizio
- **application.properties** contiene le proprietà dell'applicazione:
 - Semplice formato **chiave=valore** (es. `server.port=9090`)
 - Sono supportati anche altri formati, es. YML
- Il valore di una proprietà può essere inserito in una variabile con **@Value("\${prop.name}")**
- Le proprietà possono essere ridefinite
 - Come variabile di ambiente (**`export SERVER_PORT=9090`**)
 - Da riga di comando (**`java -jar -Dserver.port=9090 service.jar`**)

EXPLORER

- ALLEN
 - .mvn
 - .vscode
 - src
 - main
 - java/com/labisd/alien
 - bean
 - Mars.java
 - Planet.java
 - Venus.java
 - controller
 - HelloController.java
 - AlienApplication.java
 - resources
 - static
 - templates
 - application.properties
 - test
 - target
 - .gitattributes
 - .gitignore
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN

```
src > main > java > com > labisd > alien > AlienApplication.java > AlienApplication > planet()
1 package com.labisd.alien;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.beans.factory.annotation.Value;
7
8 import com.labisd.alien.bean.*;
9
10 @SpringBootApplication
11 public class AlienApplication {
12
13     @Value("${app.planet}")
14     private String planet;
15
16     public static void main(String[] args) {
17         SpringApplication.run(AlienApplication.class, args);
18     }
19
20     // @Bean vanno definiti in una classe @Configuration(@SpringBootApplication)
21     @Bean
22     public Planet planet(){
23         System.out.println("Landing planet: " + planet);
24
25         switch (planet) {
26             case "mars":
27                 return new Mars();
28             case "venus":
29                 return new Venus();
```

• Nella variabile annotata con **@Value** posso inserire la proprietà da cui verrà preso il valore da inserire nella variabile **planet**

EXPLORER

- ALIENT
- > .mvn
- > .vscode
- src
 - main
 - java/com/labisd/alien
 - bean
 - Mars.java
 - Planet.java
 - Venus.java
 - controller
 - HelloController.java
 - AlienApplication.java
 - resources
 - static
 - templates
 - application.properties**
 - test
 - target
 - .gitattributes
 - .gitignore
- OUTLINE
- TIMELINE
- JAVA PROJECTS
- MAVEN

AlienApplication.java application.properties HelloController.java Planet.java Mars.java

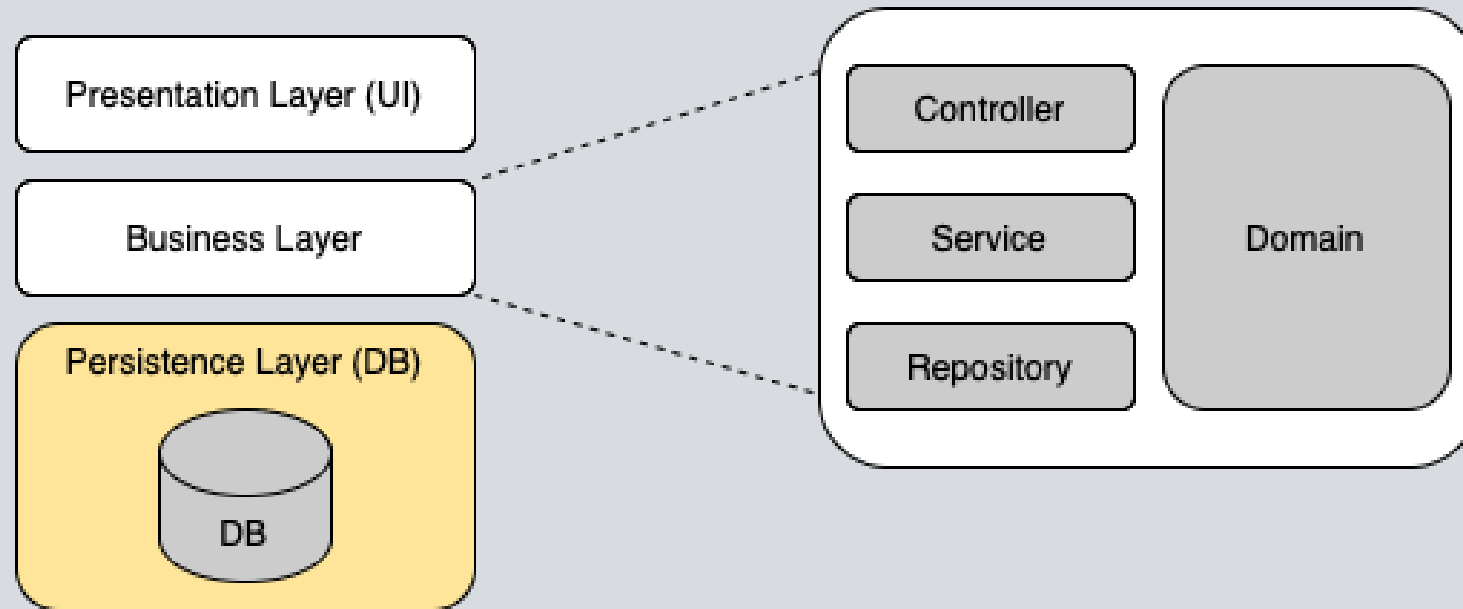
```
src > main > resources > application.properties
1  spring.application.name=alien
2
3  app.planet = mars
```

• Definisco la proprietà all'interno del file **application.properties**

Profiles

- Spesso si ha la necessità di avere configurazioni diverse in base all'ambiente di esecuzione (dev, test, prod)
 - Ad esempio, potremmo voler usare un database in memoria in fase di sviluppo
 - Un database locale in fase di test
 - Un database remoto in fase di produzione
- I profili possono essere usati per caricare file di configurazione diversi all'avvio
- È possibile ridefinire alcune proprietà presenti in **application.properties** in file a parte del tipo application-{PROFILO}.properties, es. **application-test.properties**; quelle non ridefinite ereditano il valore da application.properties
- All'avvio di un'applicazione spring, se non diversamente specificato, viene caricato il profilo "default"
- Altri profili possono essere caricati tramite la proprietà **spring.profiles.active**, es. tramite `-Dspring.profiles.active=test` o ridefinendo la variabile d'ambiente `SPRING_PROFILES_ACTIVE=test`

Architettura a Microservizi



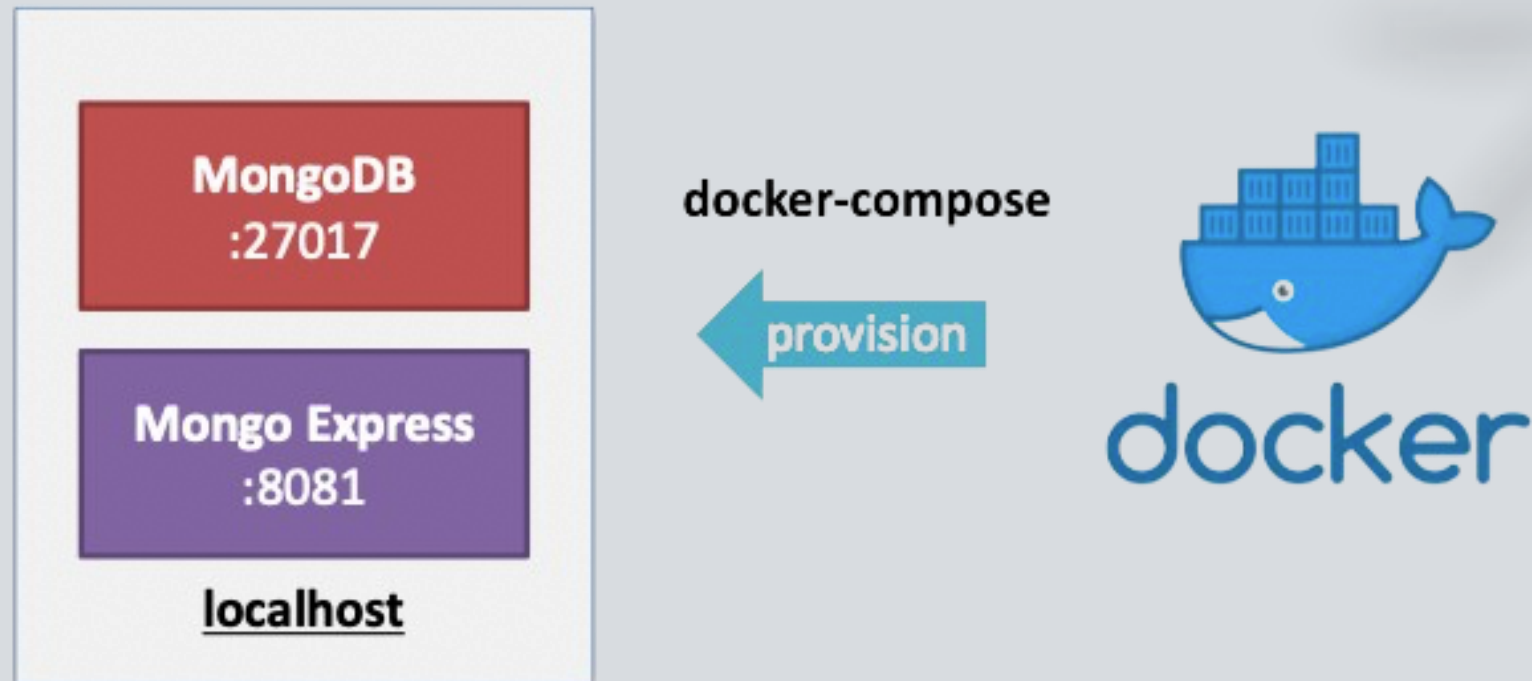
Controller: interfaccia con il client (UI), espone le API (REST)

Service: implementa la logica di business

Repository: interfaccia con il DB, espone metodi per l'interrogazione in logica OOP

Domain: contiene le entità (oggetti) per la rappresentazione e lo scambio dei dati

5. Persistence Layer con MongoDB



A step back – Install Docker

- Potete installare **docker desktop**
 - Windows: <https://docs.docker.com/desktop/setup/install/windows-install/>
 - Mac: <https://docs.docker.com/desktop/setup/install/mac-install>
- Docker desktop installa automaticamente tutto l'engine docker all'interno della vostra macchina
- Vi si potrà accedere anche tramite linea di comando

EXPLORER

- ALLEN
- src
 - main
 - java/com/labisd/alien
 - controller
 - HelloController.java
 - AlienApplication.java
 - resources
 - static
 - templates
 - application.properties
 - test
 - target
 - .gitattributes
 - .gitignore
 - docker-compose.dev.yml
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
- OUTLINE
- TIMELINE
- JAVA PROJECTS
- MAVEN

```
AlienApplication.java
docker-compose.dev.yml
HelloController.java
Planet.java
Mars.java

docker-compose.dev.yml
1  version: '3.1'
2
3  services:
4
5      mongo:
6          image: mongo
7          restart: always
8          ports:
9              - 27017:27017
10         environment:
11             MONGO_INITDB_ROOT_USERNAME: root
12             MONGO_INITDB_ROOT_PASSWORD: example
13
14         mongo-express:
15             image: mongo-express
16             restart: always
17             ports:
18                 - 8081:8081
19             environment:
20                 ME_CONFIG_MONGODB_ADMINUSERNAME: root
21                 ME_CONFIG_MONGODB_ADMINPASSWORD: example
22
23
```

HelloController.java Planet.java Mars.java

- Creiamo un file docker-compose.dev.yml per inizializzare il nostro database MongoDB e l'interfaccia web MongoExpress

Ln 23, Col 1 Spaces: 2 UTF-8 LF Compose

Docker Compose

Per creare ed avviare i container:

\$ docker-compose -f docker-compose.dev.yml up

Per stoppare ed eliminare i container:

\$ docker-compose -f docker-compose.dev.yml down

- **mongo (Database MongoDB)**

- **image: mongo:** Utilizza l'immagine ufficiale di MongoDB dal Docker Hub.
- **restart: always:** Fa sì che il container si riavvii automaticamente in caso di arresto, utile per garantire la continuità del servizio.
- **ports: 27017:27017:** Mappa la porta 27017 del container alla stessa porta sulla macchina host, rendendo MongoDB accessibile tramite localhost:27017.
- environment:
 - **MONGO_INITDB_ROOT_USERNAME:** root: Definisce il nome utente dell'amministratore di MongoDB.
 - **MONGO_INITDB_ROOT_PASSWORD:** example: Imposta la password per l'utente root di MongoDB.

- **mongo-express (Interfaccia Web per MongoDB)**

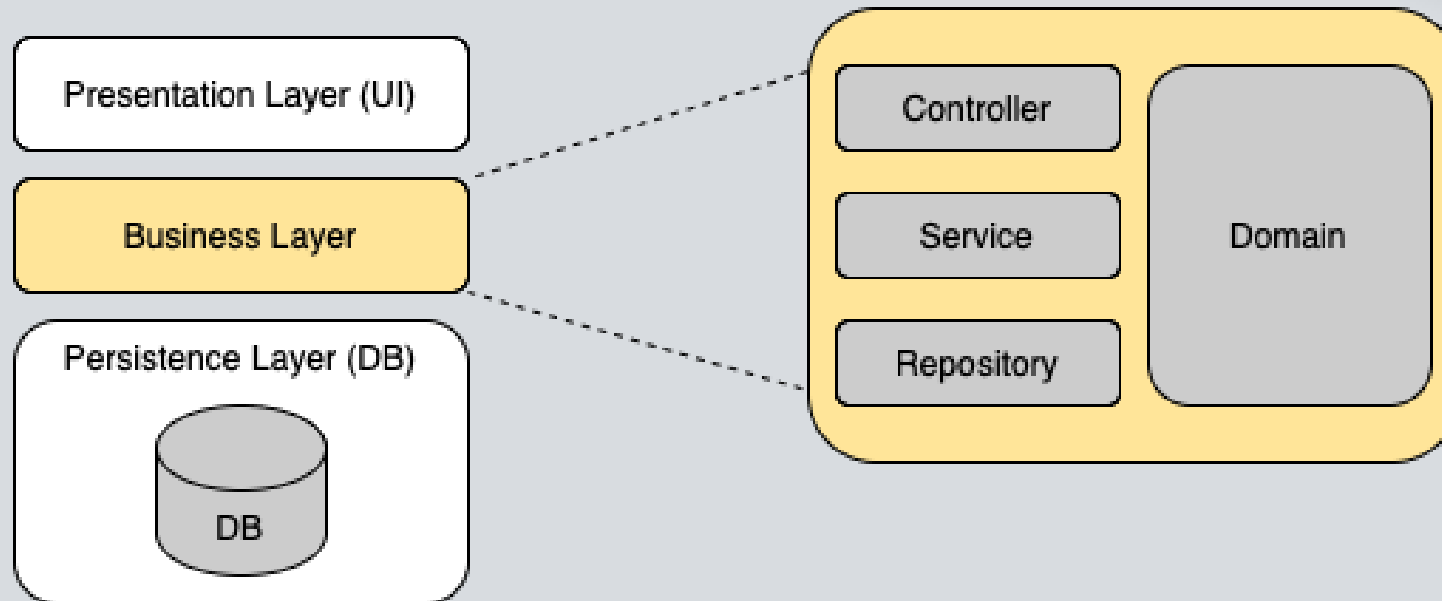
- **image: mongo-express:** Usa l'immagine ufficiale di Mongo-Express, una GUI per MongoDB.
- **ports: 8081:8081:** Mappa la porta 8081 del container alla stessa porta sull'host, rendendo l'interfaccia di Mongo-Express accessibile via browser a localhost:8081.
- environment:
 - **ME_CONFIG_MONGODB_ADMINUSERNAME:** root e **ME_CONFIG_MONGODB_ADMINPASSWORD:** example: Forniscono le credenziali per Mongo-Express, che si collega al servizio MongoDB utilizzando queste credenziali di amministratore.

Mongo Express

Databases		Database Name	+ Create Database
View	admin		Del
View	config		Del
View	local		Del

Server Status			
Hostname	c548eae50080	MongoDB Version	8.0.3
Uptime	269 seconds	Node Version	18.20.3
Server Time	Mon, 11 Nov 2024 12:01:08 GMT	V8 Version	10.2.154.26-node.37
Current Connections	3	Available Connections	838857

Architettura a Microservizi



Controller: interfaccia con il client (UI), espone le API (REST)

Service: implementa la logica di business

Repository: interfaccia con il DB, espone metodi per l'interrogazione in logica OOP

Domain: contiene le entità (oggetti) per la rappresentazione e lo scambio dei dati

6. REST API – Package Structure

```
com.space.scanner.alien
+- AlienApplication.java
+- bean
|   +- Planet.java
|   +- Mars.java
|   +- Vanus.java
+- domain
|   +- Alien.java
+- controller
|   +- HelloController.java
|   +- AlienController.java
+- service
|   +- AlientService.java
+- repository
    +- AlienRepository.java
```

consigliata

VS

```
com.space.scanner.alien
+- AlienApplication.java
+- bean
|   +- Planet.java
|   +- Mars.java
|   +- Vanus.java
+- hello
|   +- HelloController.java
+- alien
    +- Alien
    +- AlienController.java
    +- AlientService.java
    +- AlienRepository.java
```

EXPLORER

- ALIENT
- .mvn
- .vscode
- src
 - main
 - java/com/labisd/alien
 - bean
 - controller
 - domain
 - Alien.java
 - AlienApplication.java
 - resources
 - test
 - target
 - .gitattributes
 - .gitignore
 - docker-compose.dev.yml
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

OUTLINE

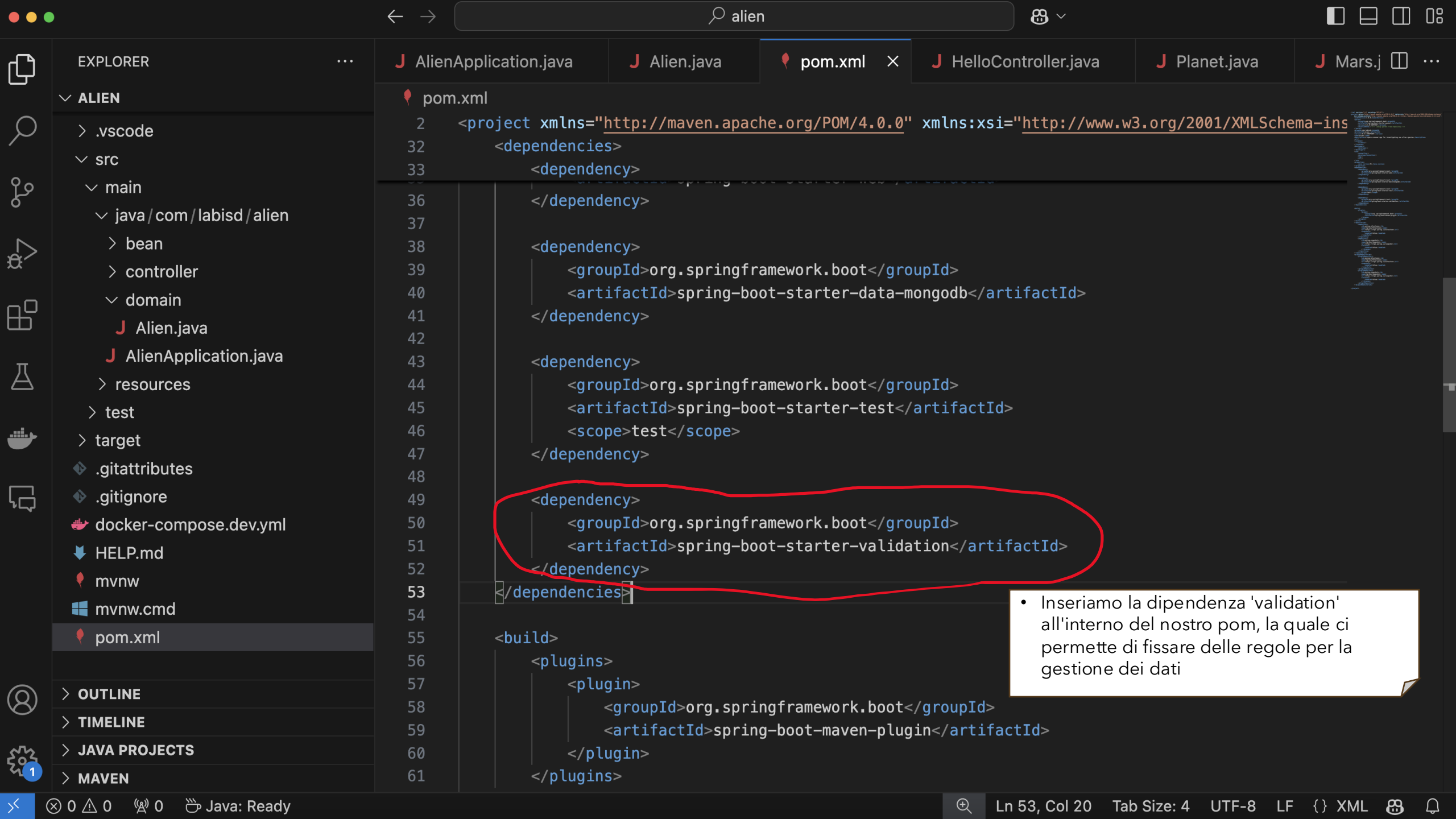
TIMELINE

JAVA PROJECTS

MAVEN

```
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-imp
29 <properties>
30 |   <java.version>21</java.version>
31 </properties>
32 <dependencies>
33 |   <dependency>
34 |     <groupId>org.springframework.boot</groupId>
35 |     <artifactId>spring-boot-starter-web</artifactId>
36 |   </dependency>
37 |
38 |   <dependency>
39 |     <groupId>org.springframework.boot</groupId>
40 |     <artifactId>spring-boot-starter-data-mongodb</artifactId>
41 |   </dependency>
42 |
43 |   <dependency>
44 |     <groupId>org.springframework.boot</groupId>
45 |     <artifactId>spring-boot-starter-test</artifactId>
46 |     <scope>test</scope>
47 |   </dependency>
48 </dependencies>
49
50 <build>
51 |   <plugins>
52 |     <plugin>
53 |       <groupId>org.springframework.boot</groupId>
54 |       <artifactId>spring-boot-maven-plugin</artifactId>
55 |     </plugin>
56 </plugins>
```

• Inseriamo la dipendenza di MongoDB all'interno del nostro progetto nel file pom.xml



```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

• Inseriamo la dipendenza 'validation' all'interno del nostro pom, la quale ci permette di fissare delle regole per la gestione dei dati

EXPLORER

- ALIENT
- .vscode
- src
 - main
 - java/com/labisd/alien
 - bean
 - controller
 - domain
 - Alien.java
 - AlienApplication.java
 - resources
 - test
 - target
 - .gitattributes
 - .gitignore
 - docker-compose.dev.yml
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
- OUTLINE
- TIMELINE
- JAVA PROJECTS
- MAVEN

```
src > main > java > com > labisd > alien > domain > Alien.java > Alien
6 import jakarta.validation.constraints.NotNull;
7 import jakarta.validation.constraints.Null;
8 import jakarta.validation.constraints.Size;
9
10 @Document
11 public class Alien {
12
13     @Id
14     private String id;
15
16     @NotNull
17     @Size(min=3, message="Name should have atleast 3 characters")
18     private String name;
19
20     @NotNull
21     @Size(max=10, message="Name should have no more than 10 characters")
22     private String race;
23
24     @Null
25     private String planet;
26
27     public Alien() {
28     }
29
30     @Override
31     public String toString() {
32         return String.format(
33             format:"Alien[id=%s, name='%s', race='%s', planet='%s'",
34             id, name, race, planet);
35     }

```

Popoliamo il domain creando la prima **entità** del nostro DB

- **@Document** mappa la classe ad un'entità (documento, in Mongo) nel DB
- **@NotNull, @Size, @Null** verranno usati per la validazione automatica
- Le entità del dominio vengono create (e popolate dal framework tramite setter) per lo scambio dati in etrambe le direzioni tra il il DB, i layer e i client REST

EXPLORER

- ALIENT
- > .mvn
- > .vscode
- src
 - main
 - java/com/labisd/alien
 - bean
 - controller
 - AlienController.java
 - HelloController.java
 - domain
 - Alien.java
 - repository
 - AlienRepository.java
 - service
 - AlienService.java
 - AlienApplication.java
 - resources
 - test
 - target
 - .gitattributes
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN

```

src > main > java > com > labisd > alien > controller > AlienController.java > AlienController
17 import org.springframework.web.bind.annotation.RestController;
18
19 import com.labisd.alien.service.AlienService;
20 import com.labisd.alien.domain.Alien;
21
22 @RestController
23 @RequestMapping("/aliens")
24 public class AlienController {
25     @Autowired
26     AlienService alienService;
27
28     @GetMapping("/add")
29     public @ResponseBody String addAlien (
30         @RequestParam String name,
31         @RequestParam String race) {
32
33         alienService.addAlien(name, race);
34         return "Saved";
35     }
36
37     @GetMapping("")
38     public @ResponseBody List<Alien> getAliens (
39         @RequestParam Optional<String> race) {
40
41         if (race.isPresent())
42             return alienService.getAliens(race.get());
43         else
44             return alienService.getAliens();
45     }
46

```

Definiamo il controller che gestirà le chiamate REST

- **@ResponseBody**: per codificare l'oggetto (entità o lista di entità) in oggetto JSON
- **@RequestParam**: per associare i parametri della richiesta (?race=marziano) ad un parametro del metodo (Optional<String> race)

EXPLORER

- ALLEN
- > .mvn
- > .vscode
- src
 - main
 - java/com/labisd/alien
 - bean
 - controller
 - AlienController.java
 - HelloController.java
 - domain
 - Alien.java
 - repository
 - AlienRepository.java
 - service
 - AlienService.java
 - AlienApplication.java
 - resources
 - test
 - target
 - .gitattributes
- OUTLINE
- TIMELINE
- JAVA PROJECTS
- MAVEN

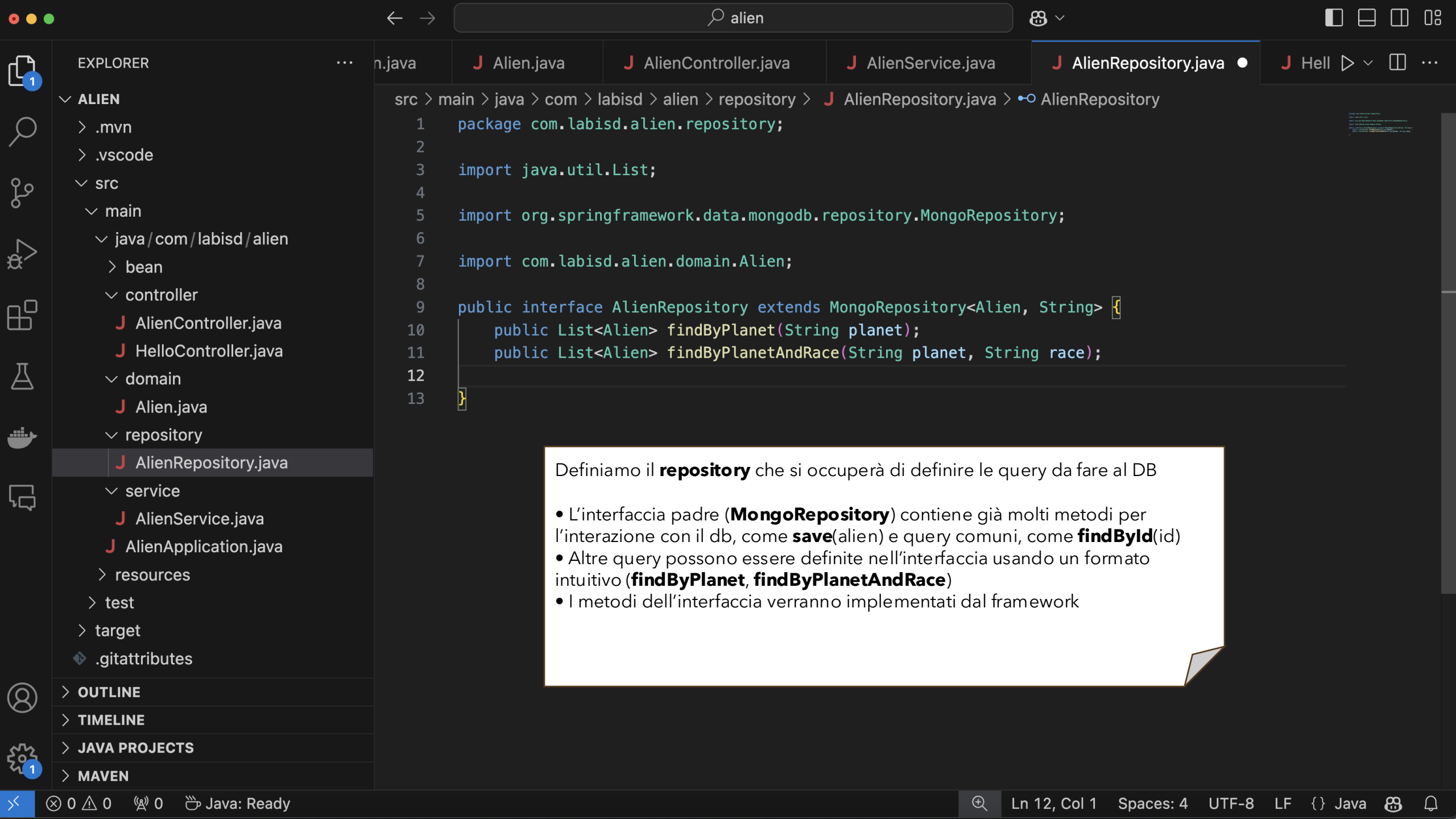
```

src > main > java > com > labisd > alien > service > AlienService.java > AlienService
3  import java.util.List;
4
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.stereotype.Service;
7
8  import com.labisd.alien.bean.Planet;
9  import com.labisd.alien.domain.Alien;
10 import com.labisd.alien.repository.AlienRepository;
11
12
13 @Service
14 public class AlienService {
15     @Autowired
16     private AlienRepository alienRepository;
17
18     @Autowired
19     Planet planet;
20
21     public void addAlien(String name, String race) {
22         Alien a = new Alien();
23         a.setName(name);
24         a.setRace(race);
25         a.setPlanet(planet.getName());
26         alienRepository.save(a);
27     }
28
29     public List<Alien> getAliens() {
30         return alienRepository.findByPlanet(planet.getName());
31     }
32

```

Definiamo il **service** che si occuperà di aggiornare il DB secondo i dati ricevuti dal controller

- Planet** e **AlienRepository** sono **@Autowired** in quanto vengono passati tramite **DI** da SpringBoot stessa, quindi si useranno le istanze attive durante l'esecuzione



src > main > java > com > labisd > alien > repository > AlienRepository.java > AlienRepository

```
1 package com.labisd.alien.repository;
2
3 import java.util.List;
4
5 import org.springframework.data.mongodb.repository.MongoRepository;
6
7 import com.labisd.alien.domain.Alien;
8
9 public interface AlienRepository extends MongoRepository<Alien, String> {
10     public List<Alien> findByPlanet(String planet);
11     public List<Alien> findByPlanetAndRace(String planet, String race);
12
13 }
```

Definiamo il **repository** che si occuperà di definire le query da fare al DB

- L'interfaccia padre (**MongoRepository**) contiene già molti metodi per l'interazione con il db, come **save**(alien) e query comuni, come **findById**(id)
- Altre query possono essere definite nell'interfaccia usando un formato intuitivo (**findByPlanet**, **findByPlanetAndRace**)
- I metodi dell'interfaccia verranno implementati dal framework

EXPLORER

- ALLEN
- src
 - main
 - java/com/labisd/alien
 - service
 - AlienService.java
 - AlienApplication.java
 - resources
 - static
 - templates
 - application.properties**
 - test
 - target
 - .gitattributes
 - .gitignore
 - docker-compose.dev.yml
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
- OUTLINE
- TIMELINE
- JAVA PROJECTS
- MAVEN

AlienService.java AlienRepository.java application.properties

```
src > main > resources > application.properties
1  spring.application.name=alien
2
3  app.planet = mars
4
5  # Tomcat configuration
6  server.port = 9090
7  # Mongo connection
8  spring.data.mongodb.authentication-database=admin
9  spring.data.mongodb.username=root
10 spring.data.mongodb.password=example
11 spring.data.mongodb.port=27017
12 spring.data.mongodb.host=127.0.0.1
13 spring.data.mongodb.database=space_scanner
14
```

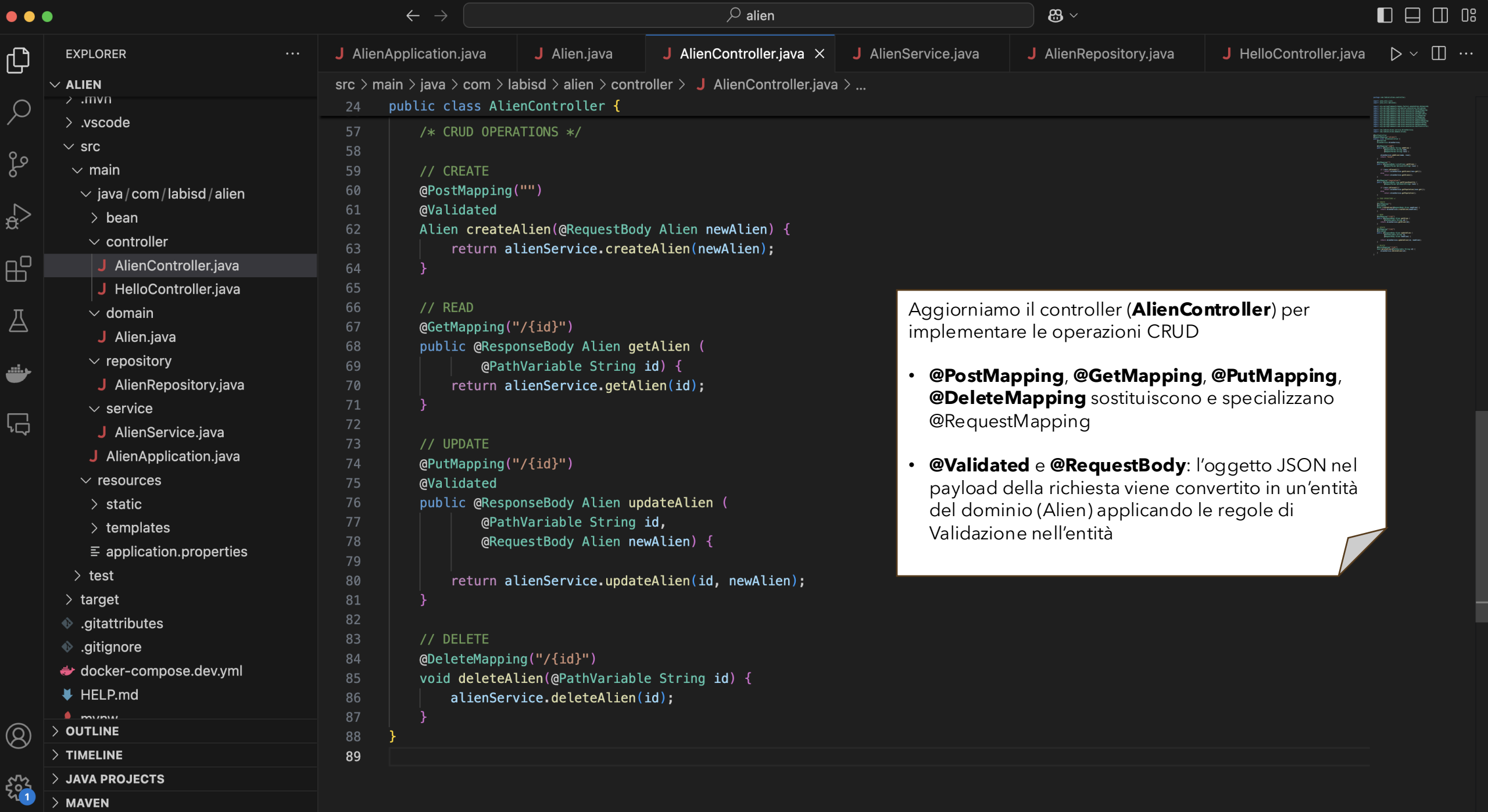
- Modifichiamo il file **application.properties** per connettere l'applicazione al DB
- Il DB è già in esecuzione in background grazie al container docker che abbiamo definito prima

Test dell'interfaccia REST

- indirizzo: `http://localhost:9090`
 - **GET /**
 - Greetings from Red Planet!
 - **GET / aliens / add?name=Urza&race=martian**
 - Saved
 - **GET / aliens / add?name=Gin&race=vulcan**
 - Saved
 - **GET / aliens**
 - [{"id":"6733266d92f8d010502d80c4","name":"Urza","race":"martian","planet":"mars"}
{"id":"673326b492f8d010502d80c5","name":"Gin","race":"vulcan","planet":"mars"}]
 - **GET / aliens?race=vulcan**
 - [{"id":"673326b492f8d010502d80c5","name":"Gin","race":"vulcan","planet":"mars"}]

7. RESTful APIs – CRUD Operations

- **REST** sta per **RE**presentational **S**tate **T**ransfer
- L'astrazione principale in REST è la **risorsa** (es. alien, population)
- Evitare quindi **azioni** (es. add)
 - usare **nomi**, non **verbi** per definire le API
- Ogni risorsa ha un **URI (Uniform Resource Identifier)**
 - aliens
 - aliens/5c492986a21e442781894b96 (id)
- Ogni risorsa ha una sua rappresentazione (XML, HTML, JSON, TEXT) che ne identifica lo **stato attuale**
- REST utilizza **verbi di HTTP** per codificare le **operazioni** sulle risorse
 - **POST: Crea** una risorsa (**C**REATE)
 - **GET: Leggi** una risorsa (**R**EAD)
 - **PUT: Aggiorna** una risorsa (**U**PNDATE)
 - **DELETE: Elimina** una risorsa (**D**ELETE)



Aggiorniamo il controller (**AlienController**) per implementare le operazioni CRUD

- **@PostMapping**, **@GetMapping**, **@PutMapping**, **@DeleteMapping** sostituiscono e specializzano @RequestMapping
- **@Validated** e **@RequestBody**: l'oggetto JSON nel payload della richiesta viene convertito in un'entità del dominio (Alien) applicando le regole di Validazione nell'entità

EXPLORER

- ALIENT
- .mvn
- .vscode
- src
 - main
 - java/com/labisd/alien
 - bean
 - controller
 - AlienController.java
 - HelloController.java
 - domain
 - Alien.java
 - repository
 - AlienRepository.java
 - service
 - AlienService.java
 - AlienApplication.java
 - resources
 - static
 - templates
 - application.properties
 - test
 - target
 - .gitattributes
 - .gitignore
 - docker-compose.dev.yml
 - HELP.md
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN

```
src > main > java > com > labisd > alien > service > AlienService.java > ...
14 public class AlienService {
37     public Long getPopulation(String race) {
39     }
40
41     public Long getPopulation() {
42         return alienRepository.countByPlanet(planet.getName());
43     }
44
45     public Alien createAlien(Alien newAlien) {
46         newAlien.setPlanet(planet.getName());
47         return alienRepository.save(newAlien);
48     }
49
50     public Alien getAlien(String id) {
51         return alienRepository.findByPlanetAndIdQuery(planet.getName(), id);
52     }
53
54     public Alien updateAlien(String id, Alien newAlien) {
55         return alienRepository.findByPlanetAndId(planet.getName(), id)
56             .map(alien -> {
57                 alien.setName(newAlien.getName());
58                 alien.setRace(newAlien.getRace());
59                 return alienRepository.save(alien);
60             })
61             .orElseGet(() -> {
62                 System.out.println(x:"put new with specified ID");
63                 newAlien.setId(id);
64                 newAlien.setPlanet(planet.getName());
65                 return alienRepository.save(newAlien);
66             });
67     }
68
69     public void deleteAlien(String id) {
70         alienRepository.deleteById(id);
71     }
72 }
73
```

Aggiorniamo il service (**AlienService**) per gestire le operazioni CRUD

EXPLORER

- ALLEN
- .mvn
- .vscode
- src
 - main
 - java / com / labisd / alien
 - bean
 - controller
 - AlienController.java
 - HelloController.java
 - domain
 - Alien.java
 - repository
 - AlienRepository.java
 - service
 - AlienService.java
 - AlienApplication.java
 - resources
 - static
 - templates
 - application.properties
 - test
 - target
 - .gitattributes
 - .gitignore
 - docker-compose.dev.yml
 - HELP.md
 - mvnw
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN

```

src > main > java > com > labisd > alien > repository > AlienRepository.java > ...
1  package com.labisd.alien.repository;
2
3  import java.util.List;
4  import java.util.Optional;
5
6  import org.springframework.data.mongodb.repository.MongoRepository;
7  import org.springframework.data.mongodb.repository.Query;
8
9  import com.labisd.alien.domain.Alien;
10
11 public interface AlienRepository extends MongoRepository<Alien, String> {
12     public List<Alien> findByPlanet(String planet);
13     public List<Alien> findByPlanetAndRace(String planet, String race);
14
15     public Long countByPlanet(String planet);
16     public Long countByPlanetAndRace(String planet, String race);
17
18     @Query("{planet: ?0, id: ?1}")
19     public Alien findByPlanetAndIdQuery(String planet, String id);
20     public Optional<Alien> findByPlanetAndId(String planet, String id);
21 }
22

```

Aggiorniamo il repository (**AlienRepository**) con le nuove query

- Oltre ad usare il formato del nome dei metodi, è possibile usare **@Query("query")** e creare query nel linguaggio specifico del Database (es. SQL per MySQL)
 - in questo caso opportuni oggetti JSON per interrogare MongoDB
- Oltre a findBy*, anche **countBy*** può essere usato per creare query di conteggio

@Query: è un'annotazione di Spring Data che consente di scrivere query personalizzate. Qui permette di definire una query MongoDB invece di fare affidamento sulla generazione automatica delle query basata sul nome del metodo.

"{planet: ?0, id: ?1}": è una query MongoDB scritta in JSON-like syntax, dove:

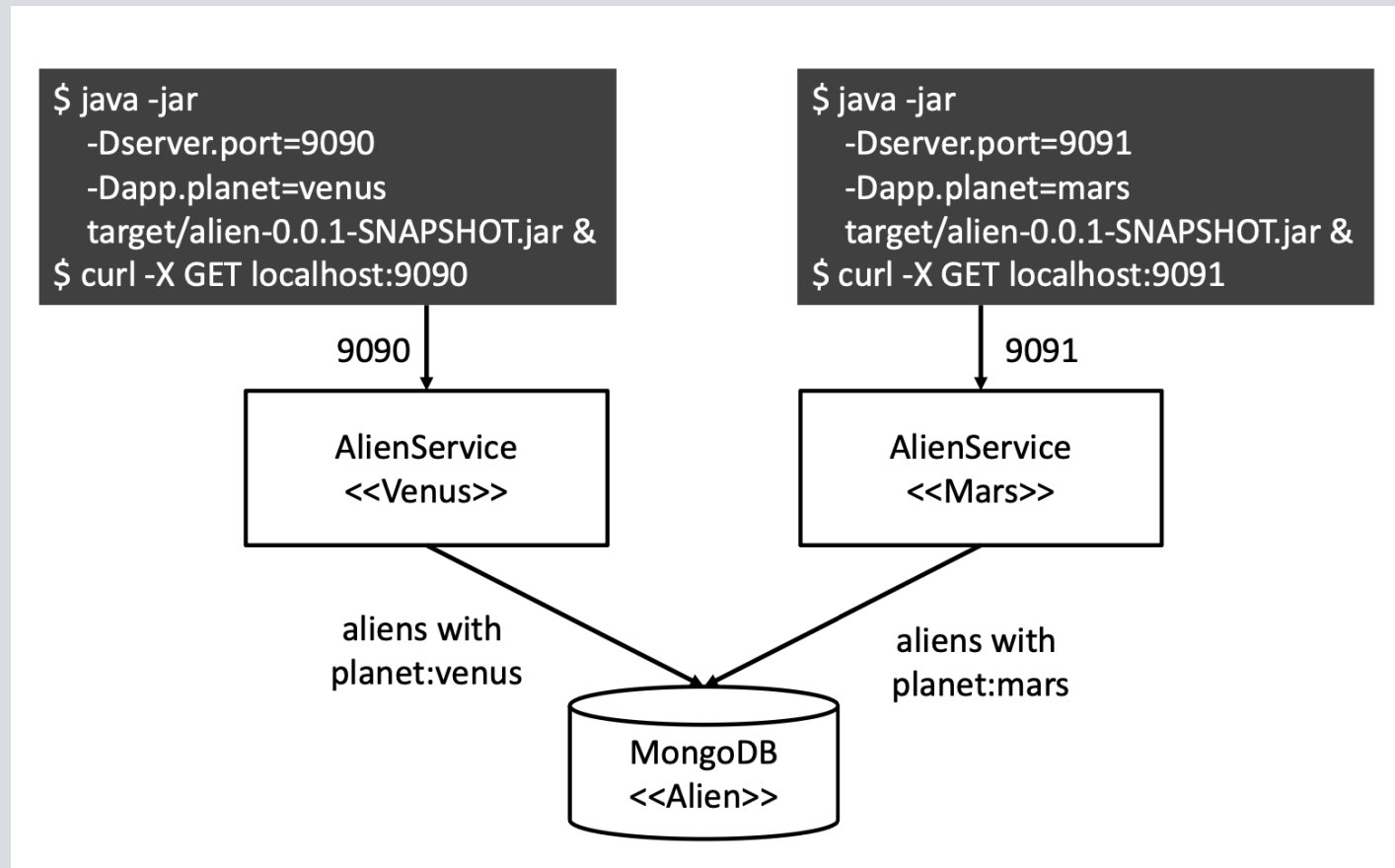
- ?0 è un placeholder per il primo parametro del metodo (planet)
- ?1 è un placeholder per il secondo parametro del metodo (id)

La query cerca i documenti nella collezione Alien in cui il campo planet corrisponde al valore del primo parametro e id corrisponde al valore del secondo parametro.

Test dell'interfaccia REST

- Usiamo **curl** da riga di comando per effettuare le chiamate
 - `curl -X POST localhost:9090/aliens -H 'Content-type:application/json' -d '{"name":"Ubaldo", "race":"terran"}'`
 - sostituire POST con il comando HTTP appropriato (GET, PUT, DELETE)
 - `-H 'Content-type:application/json'` va specificato solo se viene inviato un oggetto (-d); serve a indicare che il contenuto della richiesta (-d) è in formato JSON
- `http://localhost:9090`
 - **POST /aliens + '{"name":"Ubaldo", "race":"terran"}'**
 - `{"id":"6733361fc50b992d52b2d628","name":"Ubaldo","race":"terran","planet":"mars"}`
 - **GET /aliens/6733361fc50b992d52b2d628**
 - `{"id":"6733361fc50b992d52b2d628","name":"Ubaldo","race":"terran","planet":"mars"}`
 - **DELETE /aliens/6733361fc50b992d52b2d628**
 - `<vuoto>`
 - **GET /aliens/?race=vulcan**
 - `[{"id":"673326b492f8d010502d80c5","name":"Gin","race":"vulcan","planet":"mars"}]`

Test dei servizi: Venus e Mars



Come faccio a gestire l'accesso a più servizi?



8. API Gateway

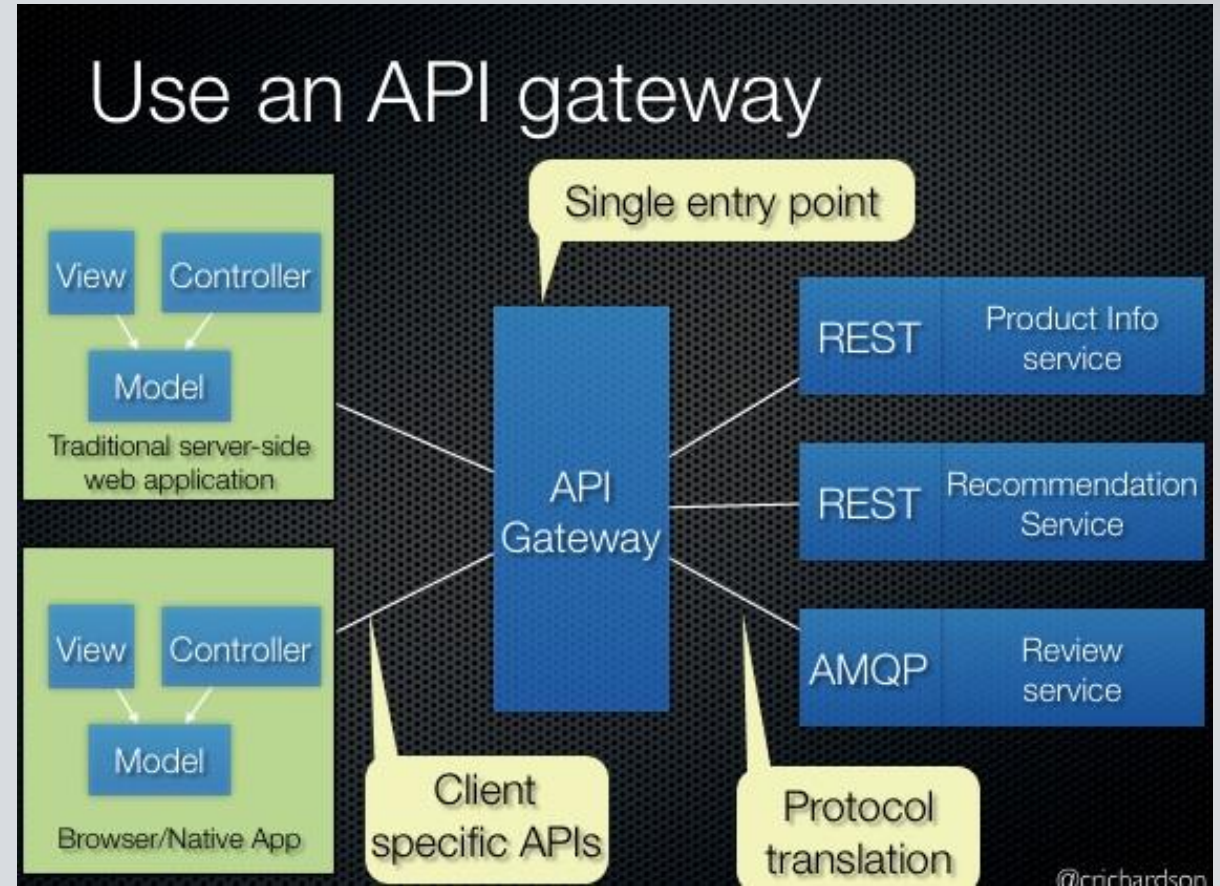
- **Contesto:** In un'architettura di microservizi, ogni servizio funziona in modo indipendente, spesso ha il proprio database e comunica su una rete utilizzando protocolli HTTP/REST, gRPC o di messaggistica. I client (ad esempio, app Web o mobili) devono interagire con più servizi per svolgere varie attività. Ad esempio, un'app mobile potrebbe dover ottenere informazioni utente, dettagli sui prodotti e raccomandazioni, ognuno dei quali è gestito da un servizio separato
- Nel nostro caso, abbiamo due servizi su porte differenti, **Mars** e **Venus**
 - Il client deve essere libero dalla configurazione dell'indirizzo, semplicemente richiede il servizio di cui ha bisogno ("atterrare" su Mars o su Venus) senza dover specificare indirizzi o porte

8. API Gateway - Problema

- **Comunicazione client complessa:** il client deve conoscere gli endpoint di ogni microservizio e chiamarli individualmente, aumentando la complessità e l'accoppiamento tra il client e i servizi backend
- **Round Trip multipli:** per eseguire il rendering di una singola pagina o schermata, un client potrebbe dover effettuare più chiamate a diversi microservizi, con conseguente aumento della latenza e maggiore sovraccarico di rete
- **Protocolli e trasformazione dei dati incoerenti:** diversi microservizi possono utilizzare protocolli, formati di dati o meccanismi di autenticazione diversi, rendendo difficile per i client gestirli tutti in modo uniforme
- **Sicurezza e autenticazione:** esporre più servizi direttamente ai client aumenta la superficie di attacco e complica la protezione degli endpoint. Ogni microservizio deve implementare i propri controlli di sicurezza, autenticazione e autorizzazione
- **Problemi trasversali:** funzionalità come la registrazione, la limitazione della velocità e la convalida delle richieste dovrebbero essere duplicate tra i servizi, portando a ridondanza e a una maggiore probabilità di implementazioni incoerenti

8. API Gateway - Soluzione

- Introduci un **API Gateway** come intermediario tra client e microservizi
- API Gateway fornisce un **singolo punto di ingresso** per tutte le richieste dei client, gestendole e indirizzandole ai microservizi appropriati in base alle necessità.

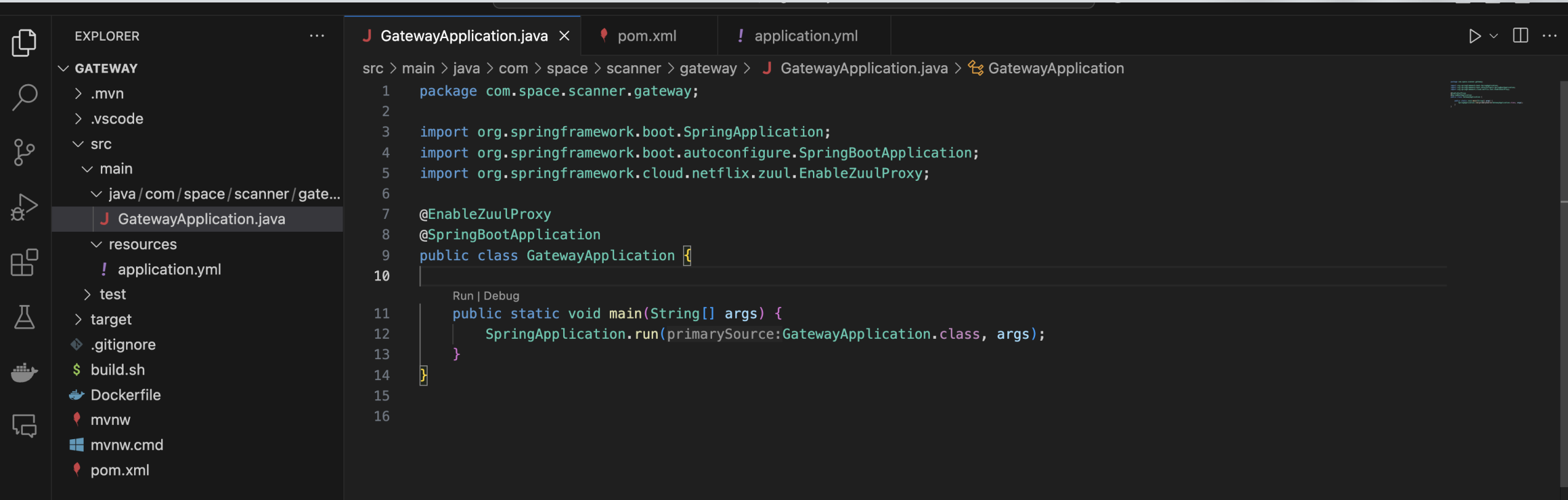


8. API Gateway - Vantaggi

- **Single Entry Point:** fornisce un singolo endpoint per tutte le richieste client, semplificando la comunicazione tramite l'astrazione della complessità sottostante dei singoli microservizi
- **Request Routing:** API Gateway indirizza le richieste ai microservizi appropriati in base al tipo di richiesta, al percorso URL o ad altri fattori, in modo che i client non debbano conoscere i microservizi specifici dietro API Gateway
- **Aggregazione delle risposte:** quando una richiesta client richiede dati da più servizi, API Gateway può effettuare chiamate parallele ai servizi pertinenti, aggregare le risposte e inviare una risposta consolidata al client. Ciò riduce il numero di round trip client-server
- **Protocollo e trasformazione dei dati:** API Gateway può gestire la traduzione del protocollo, la trasformazione del formato dei dati e persino il controllo delle versioni. Ad esempio, può convertire le risposte da diversi microservizi in un'unica risposta JSON per il client
- **Sicurezza e limitazione della velocità:** il gateway può gestire problemi trasversali come autenticazione, autorizzazione, registrazione, limitazione della velocità e memorizzazione nella cache in modo centralizzato, riducendo la ridondanza e garantendo un approccio di sicurezza coerente

API Gateway con Zuul

- Effettuate il **clone** del repository dal github del laboratorio
- Il file pom.xml è già configurato con le dipendenze necessarie per l'utilizzo di Zuul



The screenshot shows an IDE interface with the following components:

- EXPLORER:** Displays the project structure for 'GATEWAY', including folders like '.mvn', '.vscode', 'src', 'main', 'resources', 'test', 'target', and files like '.gitignore', 'build.sh', 'Dockerfile', 'mvnw', 'mvnw.cmd', and 'pom.xml'.
- FILE TABS:** Shows 'GatewayApplication.java', 'pom.xml', and 'application.yml'.
- EDITOR:** Displays the code for 'GatewayApplication.java' with the following content:

```
1 package com.space.scanner.gateway;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.zuul.EnableZuulProxy;
6
7 @EnableZuulProxy
8 @SpringBootApplication
9 public class GatewayApplication {
10
11     Run | Debug
12     public static void main(String[] args) {
13         SpringApplication.run(primarySource:GatewayApplication.class, args);
14     }
15
16 }
```


EXPLORER

- GATEWAY
 - .mvn
 - .vscode
 - src
 - main
 - java/com/space/scanner/gate...
 - GatewayApplication.java
 - resources
 - ! application.yml
 - test
 - target
 - .gitignore
 - build.sh
 - Dockerfile
 - mvnw
 - mvnw.cmd
 - pom.xml

```

src > main > resources > ! application.yml
1  server:
2    port: 8080
3
4  app:
5    service:
6      venus: http://localhost:9090
7      mars: http://localhost:9091
8
9  zuul:
10 prefix: /scanner
11 routes:
12   venus:
13     path: /venus/**
14     url: ${app.service.venus}
15   mars:
16     path: /mars/**
17     url: ${app.service.mars}

```

- Rinominare il file **application.properties** in **application.yml**
 - Il framework si occuperà di convertire le entry YML in java properties
 - Es. server.port=8080, **zuul.routes.mars.path=/venus/****
- Definisce le **regole di routing** per l'API gateway
- Le **richieste sulla 8080** verranno **reindirizzate** al servizio appropriato

OUTLINE

TIMELINE

JAVA PROJECTS

MAVEN

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

bash

The default interactive shell is now zsh.
 To update your account to use zsh, please run `chsh -s /bin/zsh`.
 For more details, please visit <https://support.apple.com/kb/HT208050>.

o Alessandros-MacBook-Air:gateway midolo\$

midolo — -bash — 80x24

```
[Alessandro-MacBook-Air:~ midolo$ curl -X GET localhost:8080/scanner/venus  
Greetings from Yellow Planet!Alessandro-MacBook-Air:~ midolo$
```

- La chiamata verrà automaticamente reindirizzata al servizio relativo:
curl -X GET localhost:8080/scanner/venus
- La risposta è infatti: **Greetings from Yellow Planet**

target — java -jar gateway-0.0.1-SNAPSHOT.jar — 80x24

```
2024-11-13 12:23:16.863 WARN 76649 --- [main] c.n.c.sources.URLConfi  
gurationSource : No URLs will be polled as dynamic configuration sources.  
2024-11-13 12:23:16.864 INFO 76649 --- [main] c.n.c.sources.URLConfi  
gurationSource : To enable URLs as dynamic configuration sources, define Sys  
tem property archaius.configurationSource.additionalUrls or make config.properti  
es available on classpath.  
2024-11-13 12:23:17.840 INFO 76649 --- [main] o.s.s.concurrent.Threa  
dPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'  
2024-11-13 12:23:19.674 INFO 76649 --- [main] o.s.c.n.zuul.ZuulFilt  
erInitializer : Starting filter initializer  
2024-11-13 12:23:19.738 INFO 76649 --- [main] o.s.b.a.e.web.Endpoint  
LinksResolver : Exposing 2 endpoint(s) beneath base path '/actuator'  
2024-11-13 12:23:20.428 INFO 76649 --- [main] o.s.b.w.embedded.tomca  
t.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''  
2024-11-13 12:23:20.436 INFO 76649 --- [main] c.s.scanner.gateway.Ga  
tewayApplication : Started GatewayApplication in 28.915 seconds (JVM running f  
or 32.854)  
2024-11-13 12:23:27.251 INFO 76649 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[lo  
calhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'  
2024-11-13 12:23:27.252 INFO 76649 --- [nio-8080-exec-1] o.s.web.servlet.Dispat  
cherServlet : Initializing Servlet 'dispatcherServlet'  
2024-11-13 12:23:27.283 INFO 76649 --- [nio-8080-exec-1] o.s.web.servlet.Dispat  
cherServlet : Completed initialization in 31 ms
```

target — java -jar -Dserver.port=9090 -Dapp.planet=venus alien-0.0.1-S...

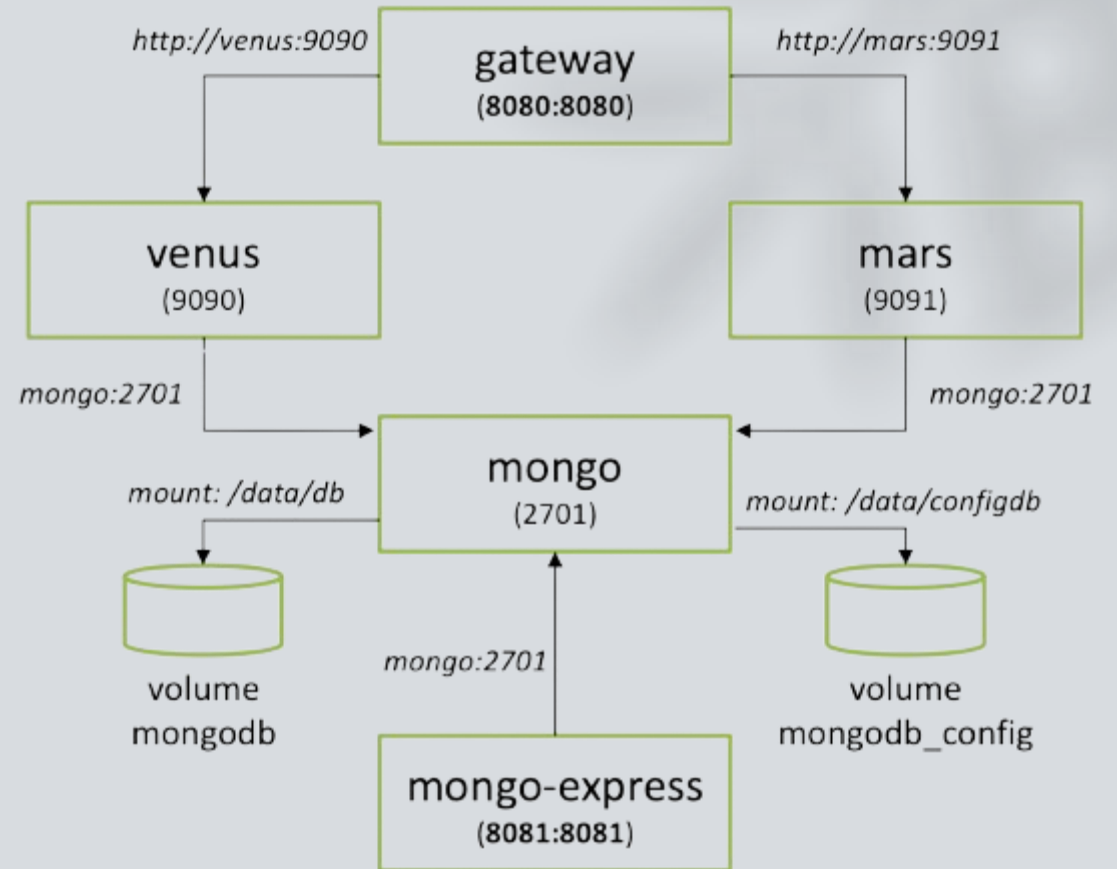
```
ssorList=[], uuidRepresentation=JAVA_LEGACY, serverApi=null, autoEncryptionSetti  
ngs=null, dnsClient=null, inetAddressResolver=null, contextProvider=null, timeou  
tMS=null}  
2024-11-13T12:23:00.438+01:00 INFO 76646 --- [alien] [127.0.0.1:27017] org.mong  
odb.driver.cluster : Monitor thread successfully connected to serv  
er with description ServerDescription{address=127.0.0.1:27017, type=STANDALONE,  
cryptd=false, state=CONNECTED, ok=true, minWireVersion=0, maxWireVersion=25, max  
DocumentSize=16777216, logicalSessionTimeoutMinutes=30, roundTripTimeNanos=13499  
5583, minRoundTripTimeNanos=0}  
Landing planet: venus  
2024-11-13T12:23:04.892+01:00 INFO 76646 --- [alien] [main] o.s.b.w.  
embedded.tomcat.TomcatWebServer : Tomcat started on port 9090 (http) with conte  
xt path '/'  
2024-11-13T12:23:04.966+01:00 INFO 76646 --- [alien] [main] com.labi  
sd.alien.AlienApplication : Started AlienApplication in 18.059 seconds (p  
rocess running for 20.06)  
2024-11-13T12:23:28.220+01:00 INFO 76646 --- [alien] [nio-9090-exec-1] o.a.c.c.  
C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispat  
cherServlet'  
2024-11-13T12:23:28.221+01:00 INFO 76646 --- [alien] [nio-9090-exec-1] o.s.web.  
servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'  
2024-11-13T12:23:28.235+01:00 INFO 76646 --- [alien] [nio-9090-exec-1] o.s.web.  
servlet.DispatcherServlet : Completed initialization in 14 ms
```

target — java -jar -Dserver.port=9091 -Dapp.planet=mars alien-0.0.1-SN...

```
rname=null, password=null}}, connectionPoolSettings=ConnectionPoolSettings{maxSi  
ze=100, minSize=0, maxWaitTimeMS=120000, maxConnectionLifeTimeMS=0, maxConnectio  
nIdleTimeMS=0, maintenanceInitialDelayMS=0, maintenanceFrequencyMS=60000, connec  
tionPoolListeners=[], maxConnecting=2}, serverSettings=ServerSettings{heartbeatF  
requencyMS=10000, minHeartbeatFrequencyMS=500, serverMonitoringMode=AUTO, server  
Listeners='', serverMonitorListeners=''}, sslSettings=SslSettings{enabled=false,  
invalidHostNameAllowed=false, context=null}, applicationName='null', compre  
ssorList=[], uuidRepresentation=JAVA_LEGACY, serverApi=null, autoEncryptionSetti  
ngs=null, dnsClient=null, inetAddressResolver=null, contextProvider=null, timeou  
tMS=null}  
2024-11-13T12:23:02.105+01:00 INFO 76648 --- [alien] [127.0.0.1:27017] org.mong  
odb.driver.cluster : Monitor thread successfully connected to serv  
er with description ServerDescription{address=127.0.0.1:27017, type=STANDALONE,  
cryptd=false, state=CONNECTED, ok=true, minWireVersion=0, maxWireVersion=25, max  
DocumentSize=16777216, logicalSessionTimeoutMinutes=30, roundTripTimeNanos=12598  
2167, minRoundTripTimeNanos=0}  
Landing planet: mars  
2024-11-13T12:23:06.398+01:00 INFO 76648 --- [alien] [main] o.s.b.w.  
embedded.tomcat.TomcatWebServer : Tomcat started on port 9091 (http) with conte  
xt path '/'  
2024-11-13T12:23:06.444+01:00 INFO 76648 --- [alien] [main] com.labi  
sd.alien.AlienApplication : Started AlienApplication in 17.911 seconds (p  
rocess running for 20.271)
```

9. Dockerize

- I container appartengono tutti alla **stessa network** di default, possono quindi comunicare tra loro (socket) tramite il **nome del servizio** e la **porta esposta**
- Viene fatto un bind delle porte esposte dai microservizi sull'host solo per i servizi che vogliamo rendere raggiungibili dall'esterno: **gateway** e **mongo-express**



Dockerfile

```
Dockerfile > FROM
1 FROM openjdk:11
2 WORKDIR /app
3 COPY target/alien-0.0.1-SNAPSHOT.jar /app
4 ENV SERVER_PORT 5000
5 EXPOSE $SERVER_PORT
6 CMD ["java", "-jar", "alien-0.0.1-SNAPSHOT.jar" ]
7
```

```
Dockerfile > ...
1 FROM openjdk:11
2 WORKDIR /app
3 COPY target/gateway-0.0.1-SNAPSHOT.jar /app
4 ENV SERVER_PORT 5000
5 EXPOSE $SERVER_PORT
6 CMD ["java", "-jar", "gateway-0.0.1-SNAPSHOT.jar" ]
7
```

```
$ cd alien
```

```
$ mvn package
```

```
$ docker build --tag space-scanner-alien:latest .
```

```
$ cd gateway
```

```
$ mvn package
```

```
$ docker build --tag space-scanner-gateway:latest .
```

Docker Compose

\$ docker-compose up
\$ docker-compose down

Nota: se non specificato, viene
usato docker-compose.yml

I volumi non verranno eliminati

```
docker-compose.yml
1  services:
2    venus:
3      image: space-scanner-alien:latest          37
4      restart: always                          38
5      #ports:                                   39
6      # - 9090:9090                             40
7      environment:                             41
8        SERVER_PORT: 9090                      42
9        APP_PLANET: venus                      43
10       SPRING_PROFILES_ACTIVE: prod           44
11       SPRING_DATA_MONGODB_HOST: mongo       45
12
13     mars:
14       image: space-scanner-alien:latest      46
15       restart: always                       47
16       #ports:                               48
17       # - 9091:9091                         49
18       environment:                          50
19         SERVER_PORT: 9091                    51
20         APP_PLANET: mars                     52
21         SPRING_PROFILES_ACTIVE: prod        53
22         SPRING_DATA_MONGODB_HOST: mongo    54
23
24     gateway:
25       image: space-scanner-gateway:latest   55
26       restart: always                       56
27       ports:                                57
28       - 8080:8080                           58
29       environment:                          59
30         SERVER_PORT: 8080                    60
31         APP_SERVICE_VENUS: "http://venus:9090"
32         APP_SERVICE_MARS: "http://mars:9091"
33
34     mongo:
35       image: mongo                          61
36       restart: always
37       #ports:
38       # - 2701:2701
39       environment:
40         MONGO_INITDB_ROOT_USERNAME: root
41         MONGO_INITDB_ROOT_PASSWORD: example
42       volumes:
43         - mongodb:/data/db
44         - mongodb_config:/data/configdb
45
46     mongo-express:
47       image: mongo-express
48       restart: always
49       ports:
50       - 8081:8081
51       environment:
52         ME_CONFIG_MONGODB_ADMINUSERNAME: root
53         ME_CONFIG_MONGODB_ADMINPASSWORD: example
54         ME_CONFIG_MONGODB_SERVER: mongo
55
56       volumes:
57         mongodb:
58         mongodb_config:
```




Università
di Catania



The End 🐫

Alessandro Midolo, PhD

Dipartimento di Matematica e Informatica

Università di Catania

alessandro.midolo@unict.it

<https://www.dmi.unict.it/amidolo/>

A.A. 2024/2025