

# Analisi e implementazione di un'architettura backend per IoT

*Lorenzo Biotti, Alessio Danesi*

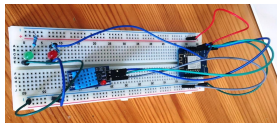
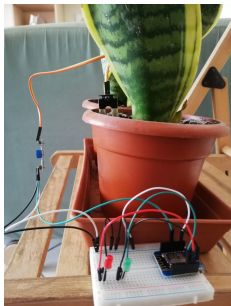
Università degli Studi di Firenze  
Scuola di Ingegneria  
Dipartimento di Ingegneria Informatica e dell'Informazione  
**Software Architecture and Methodologies**  
Prof. Enrico Vicario

June 22, 2020



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

**DINFO**  
DIPARTIMENTO DI  
INGEGNERIA  
DELL'INFORMAZIONE



- Applicazione per gestione piante e ambienti
- Monitoraggio dei valori indici di 'buona salute'
- Gestione delle reti di sensori

# Scopo del progetto



- Cambiamento architetturale con IoT
- Gestione delle reti di sensori
- Separazione delle responsabilità



- Si tratta di un **database NoSQL**
- **API REST** rendono disponibili le funzionalità di Firebase per ogni tecnologia
- **Sicurezza:** i dati immagazzinati sono replicati e sottoposti a backup continuamente. La comunicazione con i client avviene sempre in modalità crittografata tramite SSL con certificati a 2048bit
- **Costi differenziati** in base all'uso e alle capacità richiesti (free e pagamento)



- **Application server open source**, che implementa le specifiche Java EE.
- Sistema multiplatforma, interamente realizzato in Java.
- Un'estensione e un modulo che estende le funzionalità di base del server
- Il core è molto semplice e leggero

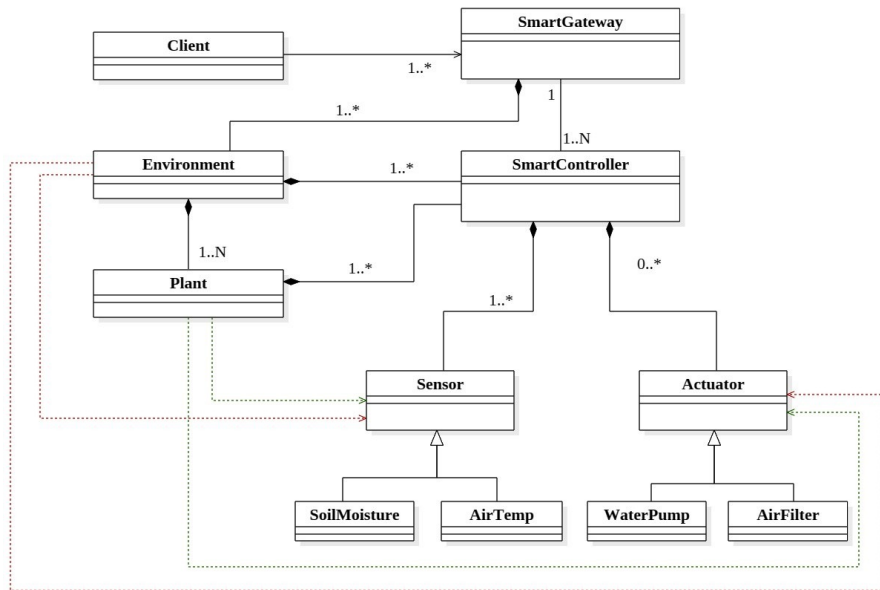


- Il broker di messaggi Kafka è stato utilizzato nella fase di gestione della ricezione dei messaggi e permette di trattare dati di diversa tipologia e provenienza all'interno dello stesso cluster.
- Kafka suddivide i **messaggi** in **topic**, e ogni topic in **partizioni**
- Un topic raggruppa messaggi dello stesso tipo e prevede la possibilità di fornire una configurazione ad hoc
- Push vs. pull: pull-based!



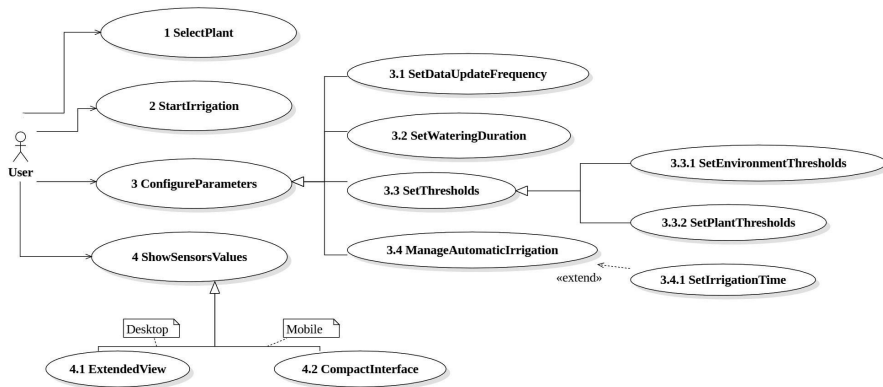
- È una piattaforma **middleware open source** per lo sviluppo di applicazioni Java, attraverso l'appoggio al relativo framework, che fornisce un servizio di **Object Relational Mapping**.
- Lo scopo principale di Hibernate è quello di fornire un mapping delle classi Java in tabelle di un database relazionale.
- Gestisce il salvataggio degli oggetti di tali classi su database.

# Analisi dei requisiti - Class diagram concettuale

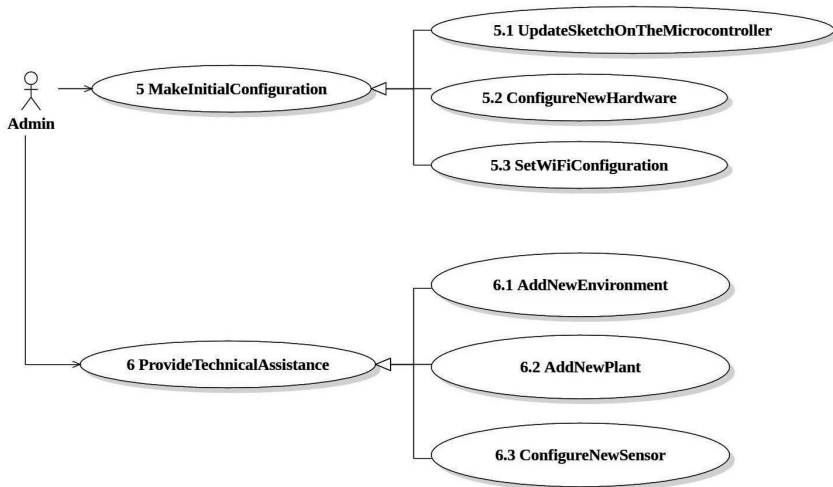




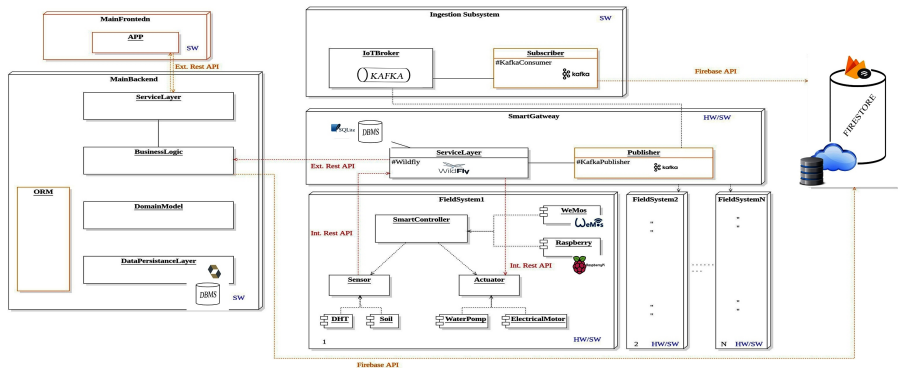
# Analisi dei requisiti - Use case diagram: user



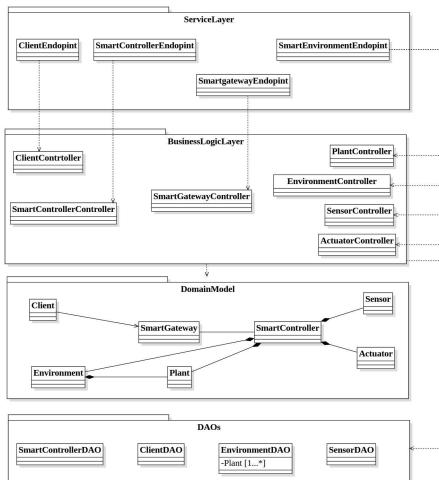
# Analisi dei requisiti - Use case diagram: admin



# Specifiche di progetto - UML Deployment diagram

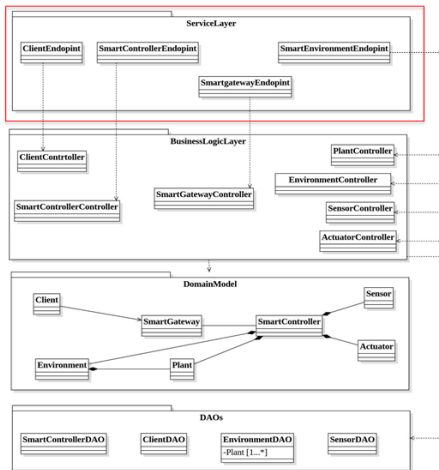


# Modello di dominio dettagliato - Main Backend



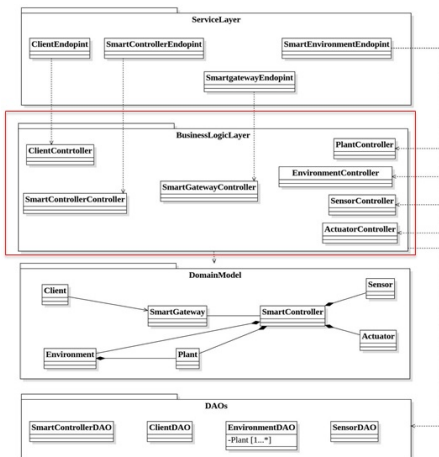
- Service Layer
- Business Logic Layer
- Domain Model
- DAOs

# Main Backend: Service Layer



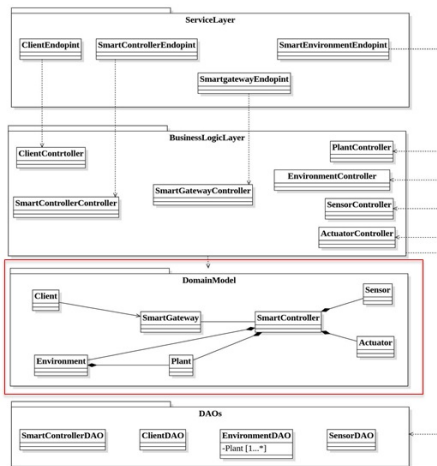
- **ClientEndpoint**
- **SmartControllerEndpoint**
- **SmartGatewayEndpoint**
- **SmartEnvironment**

# Main Backend: Business Logic layer



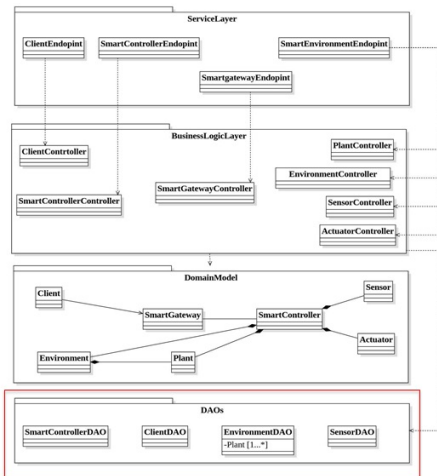
- ClientController
- SmartControllerController
- SmartGatewayController
- PlantController, EnvironmentController, SensorController e ActuatorController

# Main Backend: Domain Model



- Client
- Environment
- Plant
- SmartController
- Sensor
- Actuator
- SmartGateway

# Main Backend: DAOs

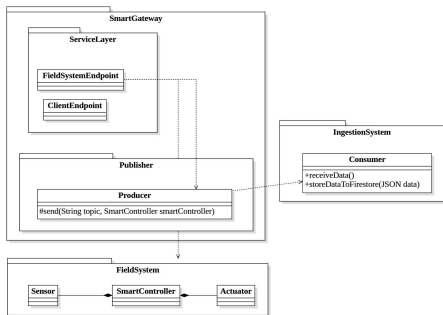


- ClientDAO
- SmartControllerDAO
- EnvironmentDAO
- SensorDAO

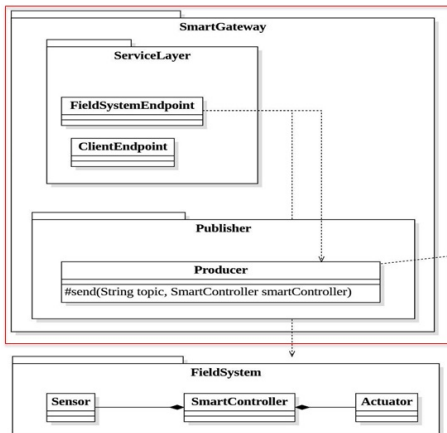


# Modello di dominio dettagliato

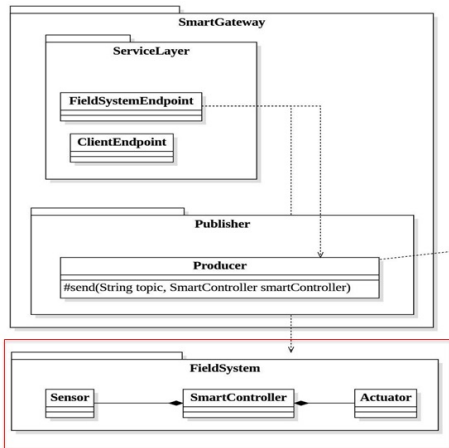
## SmartGateway-FieldSystem-IngestionSystem



- SmartGateway
- FieldSystem
- Ingestion System

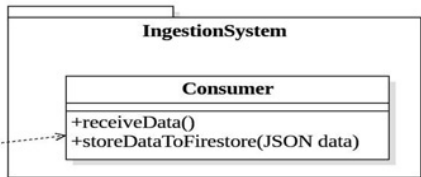


- **Service Layer**
  - **ClientEndpoint**,
  - **FieldSystemEndpoint**
- **Publisher**, costituito dalla sola classe **Producer**



- SmartController
- Sensor e Actuator

# Ingestion System



- Consumer

# Implementazione - Main Backend

```
@Path("/data")
public class DataEndpoint {

    @Inject
    private FirestoreDb firestore;

    @Path("/{name}")
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response getDataPlant(@HeaderParam("hostname") String hostname, @PathParam("name") String name) {
        try {
            Map<String,Object> obj = firestore.getLastValue(hostname, name);
            return Response.status(Status.OK).entity(obj).build();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return Response.status(Status.INTERNAL_SERVER_ERROR).build();
        }
    }
}
```



# Implementazione - Smartgateway

```
@Path("/controller")
public class SmartControllerEndpoint {

    //invoca api esposte da MainBackend
    @Path("/{idenv}/{namecontroller}")
    @PUT
    public Response addSmartController(@QueryParam("type") SmartControllerTypes type,
                                       @PathParam("idenv") Long idenv, @PathParam("namecontroller") String namecontroller) {

        CloseableHttpClient client = HttpClients.createDefault();

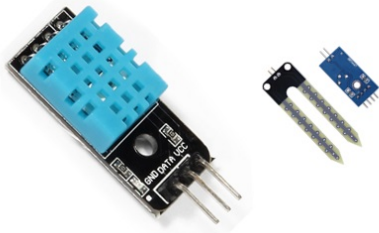
        try {
            URIBuilder uri = new URIBuilder(String.format(SmartGatewayConfig.ADD_CONTROLLER, idenv,namecontroller));
            uri = uri.setParameter("type", type.toString());
            HttpPut put = new HttpPut(uri.build());
            put.setHeader("email", SmartGatewayConfig.EMAIL_GATEWAY);
            CloseableHttpResponse response = client.execute(put);
            if (response.getStatusLine().getStatusCode()==Status.CREATED.getStatusCode())
                return Response.status(Status.CREATED).build();
        } catch (Exception e) {
            return Response.status(Status.INTERNAL_SERVER_ERROR).build();
        }

        return Response.status(Status.EXPECTATION_FAILED).build();
    }
}
```



# Implementazione - Field System

```
void makeJson() {  
    StaticJsonDocument<SIZE_BUFF> doc;  
    JsonArray sensors = doc.createNestedArray("sensors");  
    JsonObject sensors_0 = sensors.createNestedObject();  
    doc["type"] = "WemosEnv";  
    doc["frequency"] = frequency;  
    doc["name"] = myName;  
    doc["lastread"] = stamp;  
  
    sensors_0["type"] = "DHT11";  
    sensors_0["name"] = "temperatura";  
    sensors_0["threshold"] = threshold;  
    sensors_0["valueReaded"] = temp;  
    serializeJson(doc, buffJson, SIZE_BUFF);  
}
```




```
void loop() {  
  
    HTTPClient http;  
  
    http.begin("http://aledns:8080/SmartGateway/api/data");  
    http.addHeader("Content-Type", "application/json");  
  
    timeClient.update();  
  
    client = server.available();  
    rest.handle(client);  
  
    delay(frequency);  
    temp = dht.readTemperature();  
    stamp = timeClient.getFormattedTime();  
  
    makeJson();  
    int httpResponseCode = http.PUT(buffJson);  
}
```





# Implementazione - Firestore




SmartWateringPlants ▾

Database  Cloud Firestore ▾

[Dati](#) [Regole](#) [Indici](#) [Utilizzo](#)

 **Firebase**

 > [alednctest](#) > [ESP-2FD804](#)

<div> smartwateringplants</div> <div><a href="#">+ Avvia raccolta</a> Ajeje SantaMarta ale aledns@gmail.com <b>alednctest</b> &gt; alehome lore test-obj</div>	<div> alednctest</div> <div><a href="#">+ Aggiungi documento</a> <b>ESP-2FD804</b> &gt;</div>	<div> ESP-2FD804</div> <div><a href="#">+ Avvia raccolta</a> <a href="#">+ Aggiungi campo</a> <div><pre>frequency: 10000 lastread: "10:18:58" name: "ESP-2FD804" sensors   0     name: "temperatura"     threshold: 30     type: "DHT11"     valueReaded: 23.7     type: "WemosEnv"</pre></div></div>
---	--	--



# Implementazione - GitHub repository



[Code](#) [Issues 0](#) [Pull requests 3](#) [Actions](#) [Projects 0](#) [Wiki](#) [Security](#) [Insights](#)

No description, website, or topics provided.

[30 commits](#) [4 branches](#) [0 packages](#) [0 releases](#) [2 contributors](#)

[Branch: master](#) [New pull request](#) [Create new file](#) [Upload files](#) [Find file](#) [Clone or download](#)

aledns Update SmartControllerEndpoint.java		Latest commit 1a787cf 22 hours ago
IngestionSystem	Update IngestionSystem.java	11 days ago
MainBackend	Add files via upload	4 days ago
SmartGateway	Update SmartControllerEndpoint.java	22 hours ago
WemosEnv	Add files via upload	yesterday
README.md	Update README.md	4 days ago

[README.md](#)

## Swam-Code

Progetto di Alessio Danesi e Lorenzo Biotti

Per ulteriori dettagli: [Swam-code GitHub repo](#)



Si prevedono sviluppi futuri di questa architettura, con le seguenti funzionalità:

- Scalabilità hardware utilizzato
- Copia di backup delle configurazioni, anche in database locali su ogni SmartGateway
- Creazione e utilizzo di più *‘sotto-account’*

The end

*Grazie per l'attenzione!*