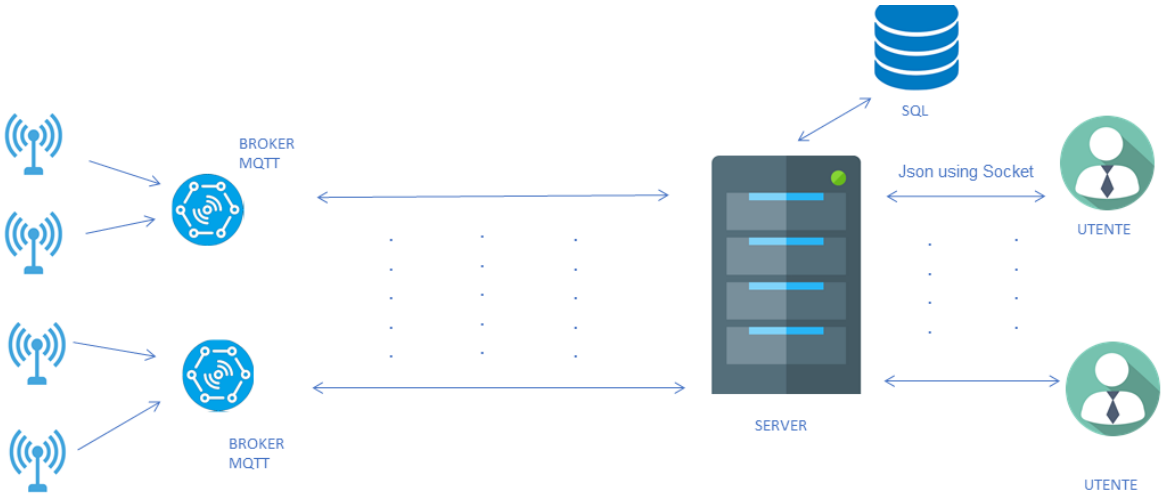
Near2U

Near2U è un sistema di gestione di ambienti smart (IoT), quali case, auto ed in generale qualsiasi ambiente connesso alla rete.
L'utente che usufruisce di questo sistema potrà vedere in tempo reale le informazioni inerenti i sensori presenti all'interno dei vari ambienti, potrà andare a comandare delle azioni agli attuatori, se presenti, e potrà configurare nuovi ambienti e modificarne di esistenti.
Di seguito viene proposta l'architettura del sistema.



Scelte Implementative

Come concordato con la professoressa è stata apportata una modifica alla comunicazione Client/Server, che non sarà più HTTP Rest, ma avverrà attraverso socket in formato JSON.
Per quanto riguarda l'implementazione si è scelto di utilizzare C++ per il server e Go per il Client. I sensori sono stati simulati attraverso dei processi separati che inviano dati al broker MQTT implementato tramite Eclipse Mosquitto.
Per le interfacce di comunicazione tra Server e broker e tra Utente e broker sono state utilizzate le seguenti librerie:

- Server side: Eclipse Paho(https://www.eclipse.org/paho/clients/c/).
- Client side: Eclipse Paho (https://github.com/eclipse/paho.mqtt.golang)

Per implementare l'interfaccia grafica è stato impiegato il binding per Qt con Go (https://github.com/therecipe/qt).
Infine per l'utilizzo dei json sono state usate le seguenti librerie(parser):

- Server side: jsoncpp(https://github.com/open-source-parsers/jsoncpp)
- Client side: libreria built-in "encoding/json".

Per la persistenza dei dati è stato utilizzato un Database SQL.

Progettazione

Di seguito vengono proposti i casi d'uso estrapolati dalla fase di Ideazione, i quali sono stati implementati in modo iterativo e completo:

Nome del caso d'uso	**UC1: Visualizza dati sensori in tempo reale**
Portata	Applicazione Near2U
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate e Interessi	- Utente: vuole una visualizzazione intuitiva e affidabile dei dati raccolti dai sensori.
Pre-condizioni	- L'utente è autenticato e identificato. - È stato configurato almeno un sensore.
Garanzia di successo	I dati vengono presentati nell'interfaccia dell'applicazione.
Scenario principale di successo	1. L'utente naviga sulla pagina di presentazione dei dati. 2. L'utente seleziona l'ambiente di cui vuole visualizzare i dati. 3. Il sistema torna all'utente i dati a intervalli regolari.
Estensioni	3a. Il sistema non trova i dati relativi all'ambiente selezionato. 1. Il sistema comunica che l'ambiente non è configurato o è inesistente.

Requisiti speciali	- Interfaccia grafica touch screen su dispositivi mobili e desktop
Elenco delle varianti tecnologiche e dei dati	
Frequenza di ripetizione	Ogni volta che un utente intende visualizzare i dati relativi a un ambiente.

Nome del caso d'uso	**UC2: Invia comandi agli attuatori**
Portata	Applicazione Near2U
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate e Interessi	- Utente: vuole inviare un comando a un attuatore di un ambiente specifico, in modo affidabile.
Pre-condizioni	- L'utente è autenticato e identificato. - È stato configurato almeno un attuatore.
Garanzia di successo	I comandi vengono inviati all'attuatore e quest'ultimo compirà l'azione specificata.
Scenario principale di successo	1. L'utente seleziona l'attività di invio di un comando. 2. L'utente seleziona l'ambiente in cui si trova l'attuatore da controllare. 3. Il sistema torna all'utente un feedback relativo all'azione compiuta.

Estensioni	3a. Il sistema non trova i dati relativi all'ambiente selezionato. 1. Il sistema comunica che l'ambiente non è configurato o è inesistente.
Requisiti speciali	- Interfaccia grafica touch screen su dispositivi mobili e desktop
Elenco delle varianti tecnologiche e dei dati	
Frequenza di ripetizione	Ogni volta che un utente intende inviare un comando ad un attuatore.

Nome del caso d'uso	**UC3: Configura ambiente**
Portata	Applicazione Near2U
Livello	Obiettivo amministratore
Attore primario	Admin
Parti interessate e Interessi	- Utente: vuole aggiungere un ambiente configurato per poter interagire con i dispositivi contenuti in esso. - Admin: vuole un'interfaccia intuitiva per il processo di configurazione
Pre-condizioni	- L'admin è autenticato e identificato.
Garanzia di successo	Un nuovo ambiente è configurato e pronto per fornire dati e ricevere comandi.

Scenario principale di successo	1. L’admin seleziona l’attività di configurazione di un nuovo ambiente. 2. L’admin inserisce il nome dell’ambiente. 3. L’admin inserisce i dispositivi presenti nell’ambiente. 4. Il sistema configura un nuovo ambiente e comunica l’avvenuta creazione.
Estensioni	1a. L'admin seleziona l’attività di configurazione di un ambiente esistente. 1. L’admin seleziona il dispositivo da configurare. 2. L’admin inserisce la nuova configurazione. 3. Il sistema informa l’admin del successo dell’operazione.
Requisiti speciali	- Interfaccia grafica touch screen su dispositivi mobili e desktop
Elenco delle varianti tecnologiche e dei dati	
Frequenza di ripetizione	Ogni volta che l’admin intende configurare un dispositivo.

Nome del caso d’uso	**UC4: Visualizza storico dei dati**
Portata	Applicazione Near2U
Livello	Obiettivo utente
Attore primario	Utente

Parti interessate e Interessi	- Utente: vuole una visualizzazione compatta di grafici relativi a dati accumulati dai sensori.
Pre-condizioni	- L'utente è autenticato e identificato. - È stato configurato almeno un sensore.
Garanzia di successo	I dati vengono presentati nell'interfaccia dell'applicazione.
Scenario principale di successo	1. L'utente naviga sulla pagina di presentazione dei dati. 2. L'utente seleziona l'ambiente di cui vuole visualizzare i dati. 3. Il sistema torna all'utente i dati relativi dell'ambiente selezionato.
Estensioni	3a. Il sistema non trova i dati relativi all'ambiente selezionato. 1. Il sistema comunica che l'ambiente non è configurato o è inesistente.
Requisiti speciali	- Interfaccia grafica touch screen su dispositivi mobili e desktop
Elenco delle varianti tecnologiche e dei dati	
Frequenza di ripetizione	Ogni volta che un utente intende visualizzare lo storico dati relativi a un ambiente.

Scelte Implementative

Server

Per l’implementazione del Server si è scelto di utilizzare come linguaggio C++ e per quanto concerne la comunicazione si sono utilizzate le socket. Lato Server all’avvio viene creato un Pool di thread, all’arrivo di una richiesta questa verrà inserita in una coda e verrà notificato un thread, il quale estrarrà la richiesta, effettuerà l’elaborazione e risponderà al server. Per far ciò all’interno della coda verranno inseriti non solo i dati della richiesta ma anche la socket, grazie alla quale ogni thread potrà rispondere al client.
Per la formattazione dei dati si è utilizzata la libreria jsoncpp che permette la manipolazione di dati in formato json.
Oltre al pool di thread all’avvio verrà eseguito un unteriore thread che avrà il compito di simulare i sensori. Il thread andrà ad iterare all’interno degli ambienti e per ogni sensore genererà dei record casuali che verranno pubblicati in un topic del broker MQTT.
Per quanto concerne il broker è stato utilizzato eclipse Mosquitto, e per il client si è utilizzata la libreria Paho Mqtt client che permette di pubblicare e sottoscriversi ai topic all’interno del broker.

Client

Il formato delle richieste utilizzato per comunicare tra client, server e broker MQTT è JSON, che viene gestito tramite una apposita funzione, buildRequest, che si occupa di costruire le richieste nel formato "funzione|dati|autenticazione". Le risposte sono nel formato "status|errore|dati" e vengono gestite lato Go tramite una interfaccia vuota che verrà trattata come mappa per poter accedere ai vari campi in modo flessibile.

Il codice client è stato separato in moduli per una maggiore coesione, ognuno operante a un diverso livello di logica applicativa.

Modulo socket.go:

Incorpora tutte le funzioni relative alla comunicazione su socket: la connessione al server, la scrittura e la lettura da socket.
Viene definita una struct che racchiude il formato della richiesta e una funzione di supporto per costruire le richieste, buildRequest() che avranno sempre lo stesso formato.
- socketConnect e socketSend funzionano in modo abbastanza lineare, la prima riceve indirizzo IP e porta del server e vi instaura una connessione, la seconda riceve i dati in formato serializzato e li scrive sulla socket.
- socketReceive legge dalla socket e memorizza su un buffer da 8192, per poi ritornare tale buffer in modo serializzato.
- socketCommunicate incorpora le funzioni precedenti ed esegue una comunicazione di tipo request/response tra client e server. Si occupa anche di costruire la richiesta e di eseguire Marshalling/Unmarshalling dei dati in formato JSON. La risposta viene salvata su una variabile di tipo map[string]interface{} per poter gestire un JSON flessibile e poter accedere ai campi d'interesse. Infine, la risposta viene ritornata al client tramite un canale di Go.

Modulo auth.go:

Comprende le funzionalità di autenticazione utente, ovvero login e registrazione.
Le funzioni accettano i parametri di autenticazione o registrazione e un canale tramite il quale verrà ritornata la risposta alla GUI.
Nella funzione Login, viene ritornato un messaggio di risposta e il token che il client utilizzerà per l'autenticazione.

Modulo client.go:

Implementa le azioni intraprese dal client a livello applicativo, includendo anche la struttura Client, che contiene i dati della sessione del client e implementa i metodi per interagire con il server.
Definisce l'istanza singleton del client e i metodi ad essa associati, che utilizzeranno come autenticazione il token ritornato dalla funzione di login.
Gestisce, inoltre, la connessione al broker MQTT e la subscribe al topic di interesse.

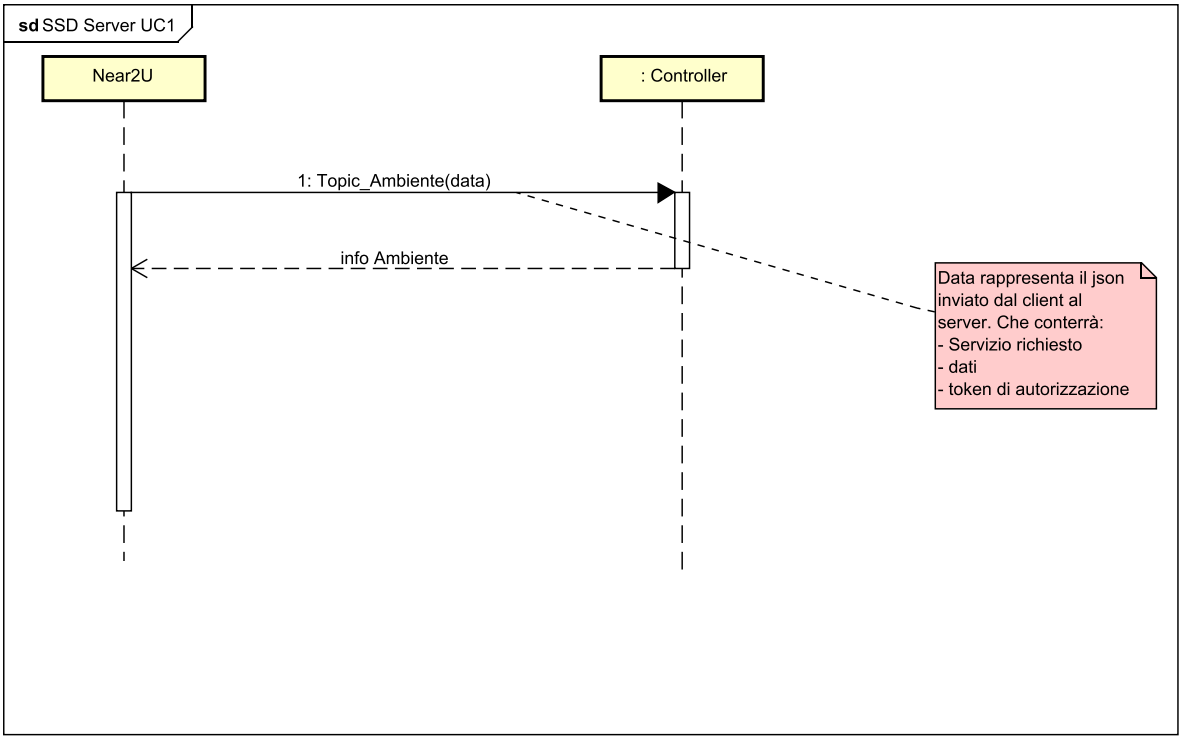
Modulo main.go:

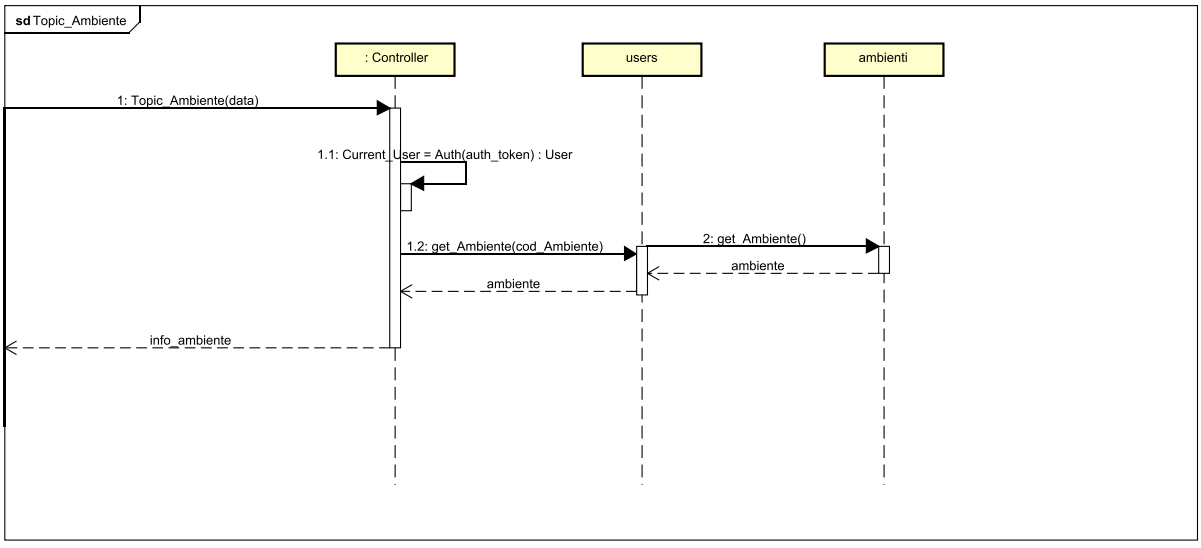
Inizializza l'applicazione e contiene l'interfaccia grafica

UC1 Visualizza dati sensori in tempo reale

Per implementare la visualizzazione dei dati dei sensori il Client invia una richiesta contenente l'ambiente di cui si vuole ricevere i dati e il server risponderà con l'indirizzo del broker mqtt al quale il client si collegherà e renderà possibile la visualizzazione all'utente dei dati dei sensori.
Di seguito vengono proposti i diagrammi di sequenza, rispettivamente per il server e per il client inerenti al caso d'uso.

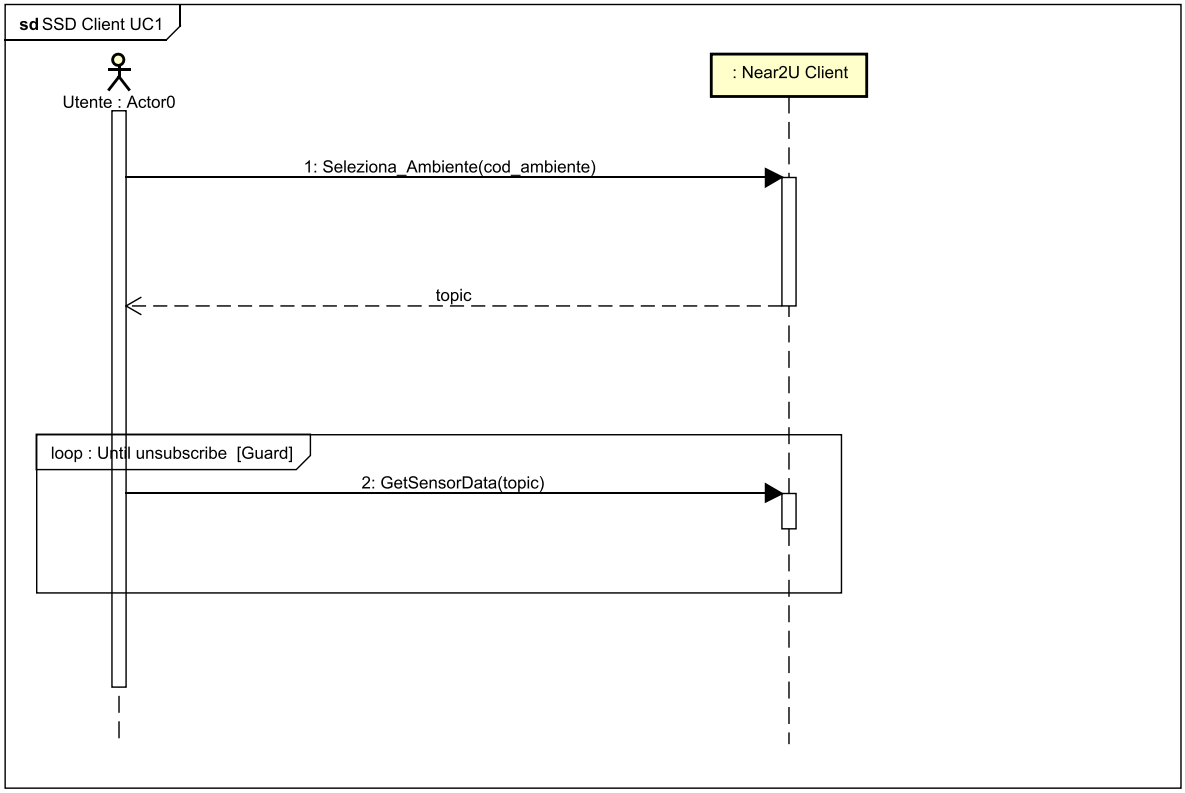
Server

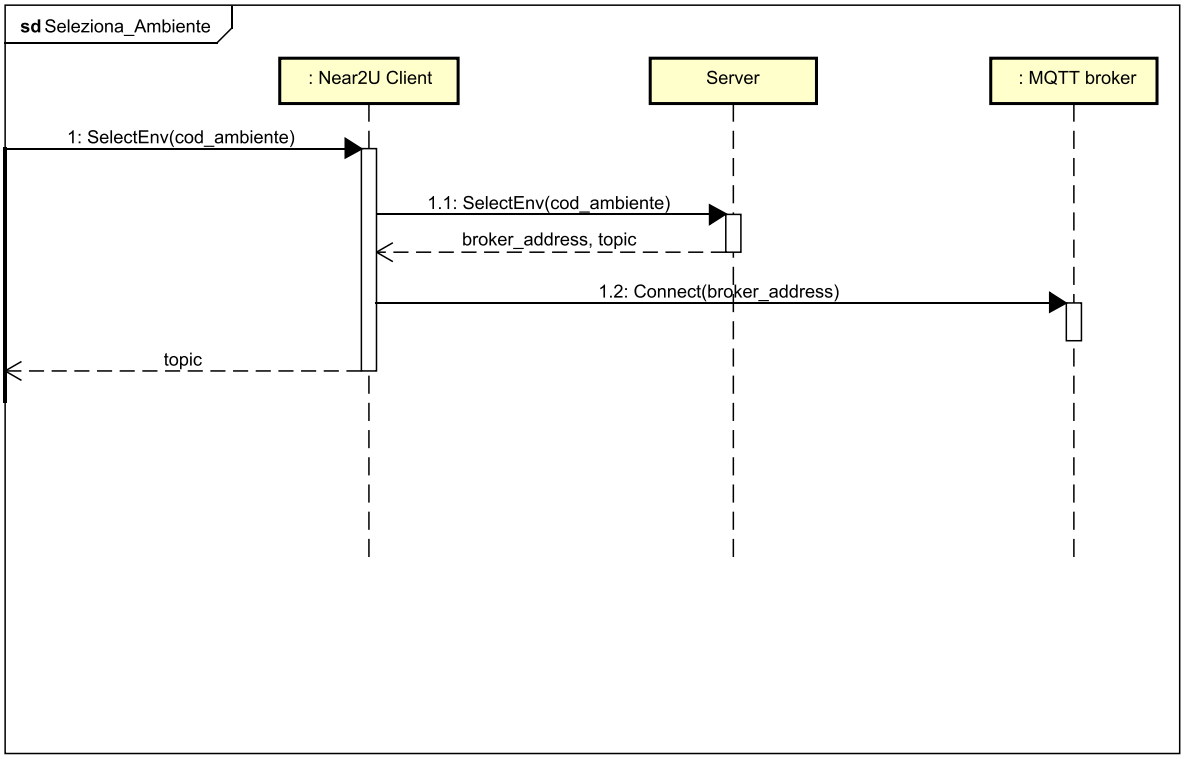


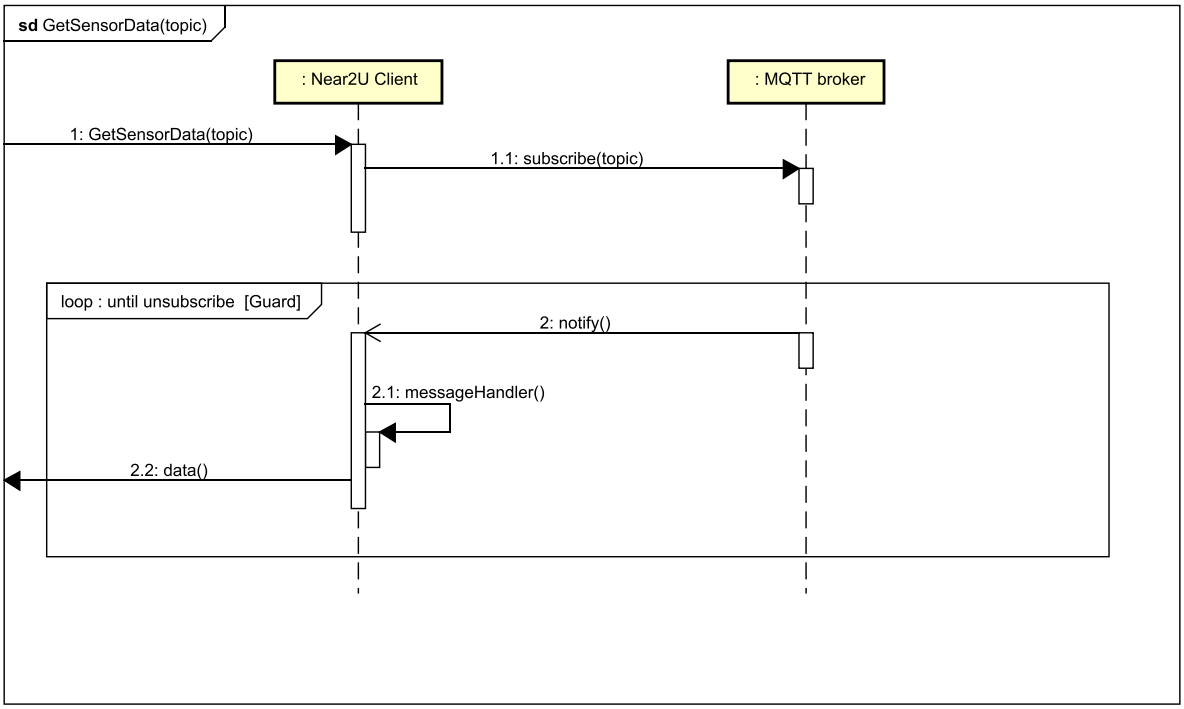


La comunicazione avverrà attraverso l'utilizzo di socket scambiando dati in formato Json.
Per quanto riguarda le richieste avranno tutte il seguente formato:
{
function:,
data: ,
auth:
}
In cui nel campo function verrà inserita la funzione da richiamare nel server, data conterrà i dati necessari alla funzione e auth conterrà il token che autorizza e identifica l'utente in maniera univoca.
La risposta del server conterrà l'indirizzo del broker (localhost:8082) e il topic dell'ambiente nel quale verranno pubblicati i dati dei sensori.

Client



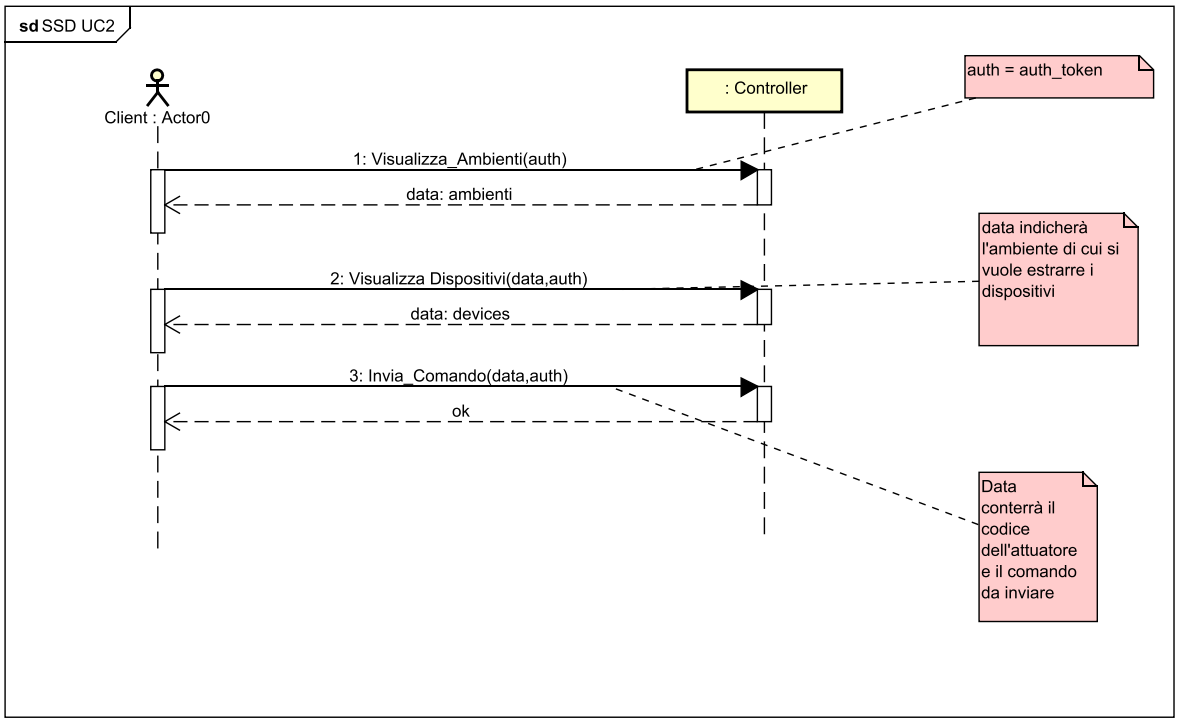

sd Seleziona_Ambiente
: Near2U Client
Server
: MQTT broker
1: SelectEnv(cod_ambiente)
1.1: SelectEnv(cod_ambiente)
broker_address, topic
1.2: Connect(broker_address)
topic


sd GetSensorData(topic)
: Near2U Client
: MQTT broker
1: GetSensorData(topic)
1.1: subscribe(topic)
loop : until unsubscribe [Guard]
2: notify()
2.1: messageHandler()
2.2: data()

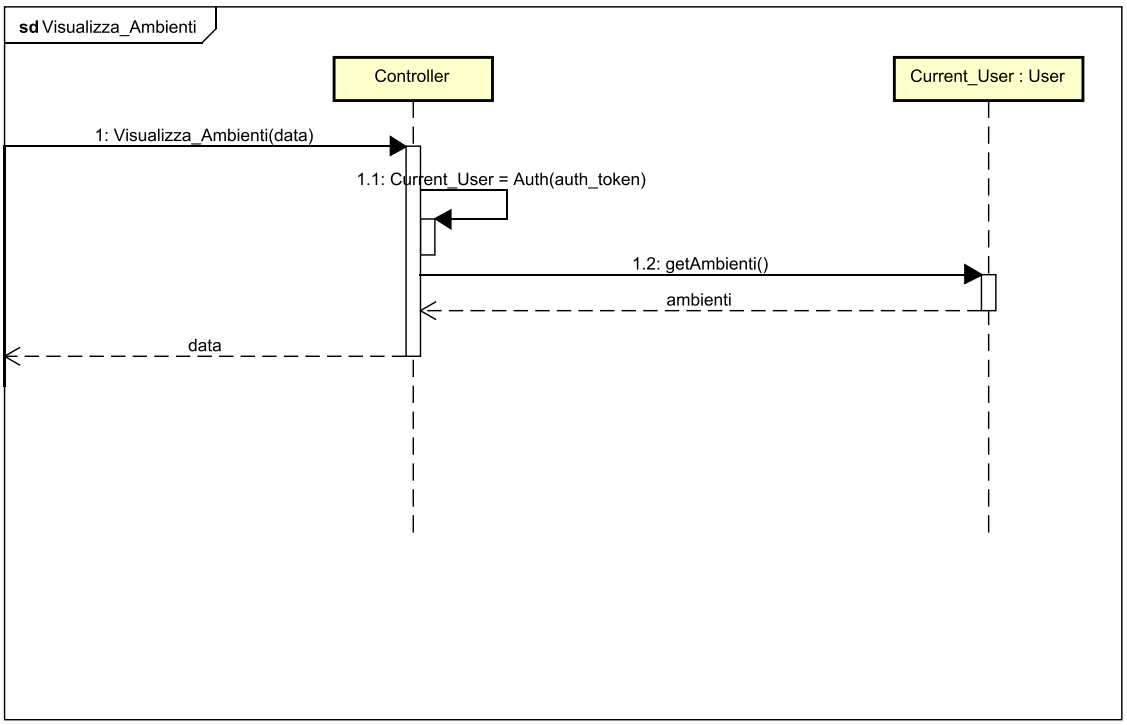
UC2: Invia comandi

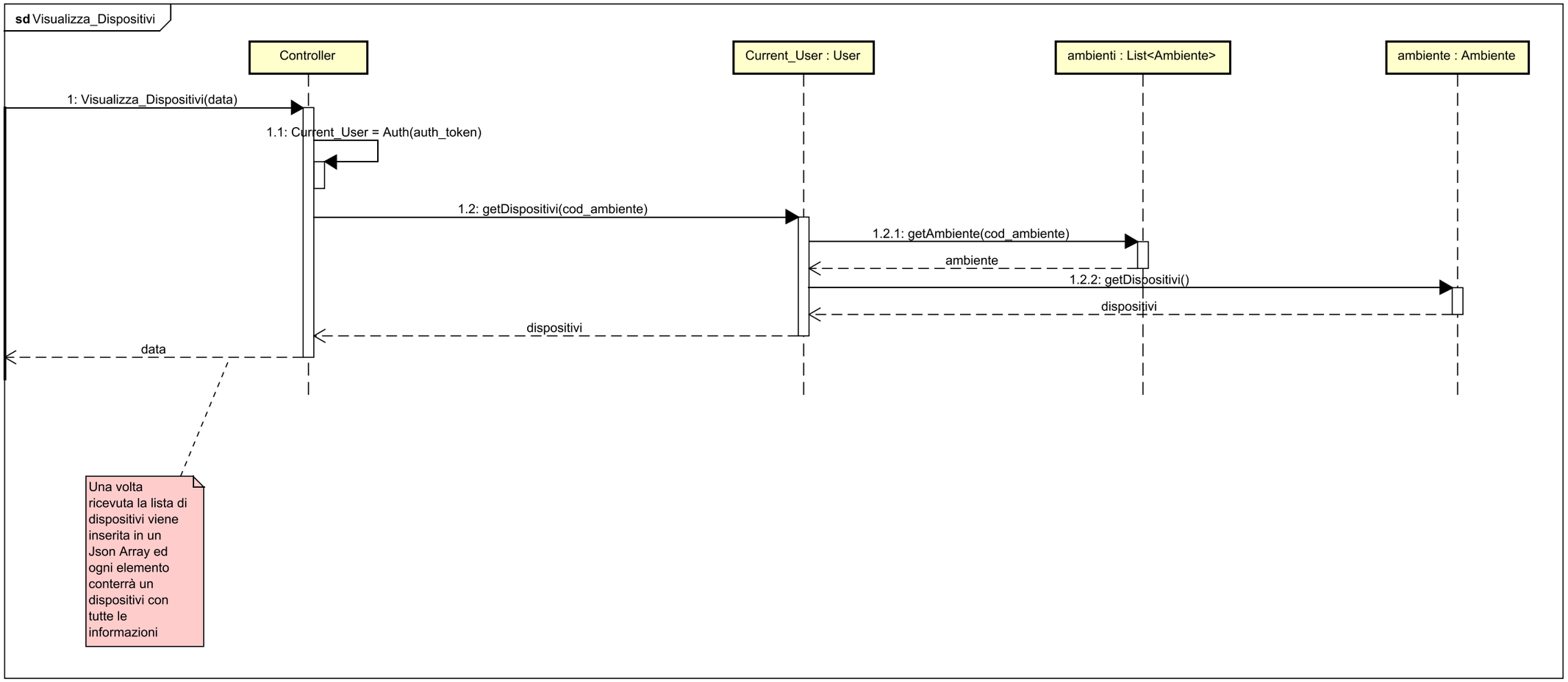
il client invierà una richiesta al server indicando di voler inviare un comando ad un determinato attuatore presente all'interno di un ambiente.
Per fare ciò il client richiederà prima gli ambienti a lui associati e successivamente gli attuatori presenti in modo da poter selezionale l'attuatore e il comando da inviare.
Il server farà un controllo sul comando e successivamente invierà il comando al topic del broker Mqtt adeguato (topic = codice ambiente + codice attuatore)
Di seguito vengono proposti i diagrammi di sequenza, rispettivamente per il server e per il client inerenti al caso d'uso.

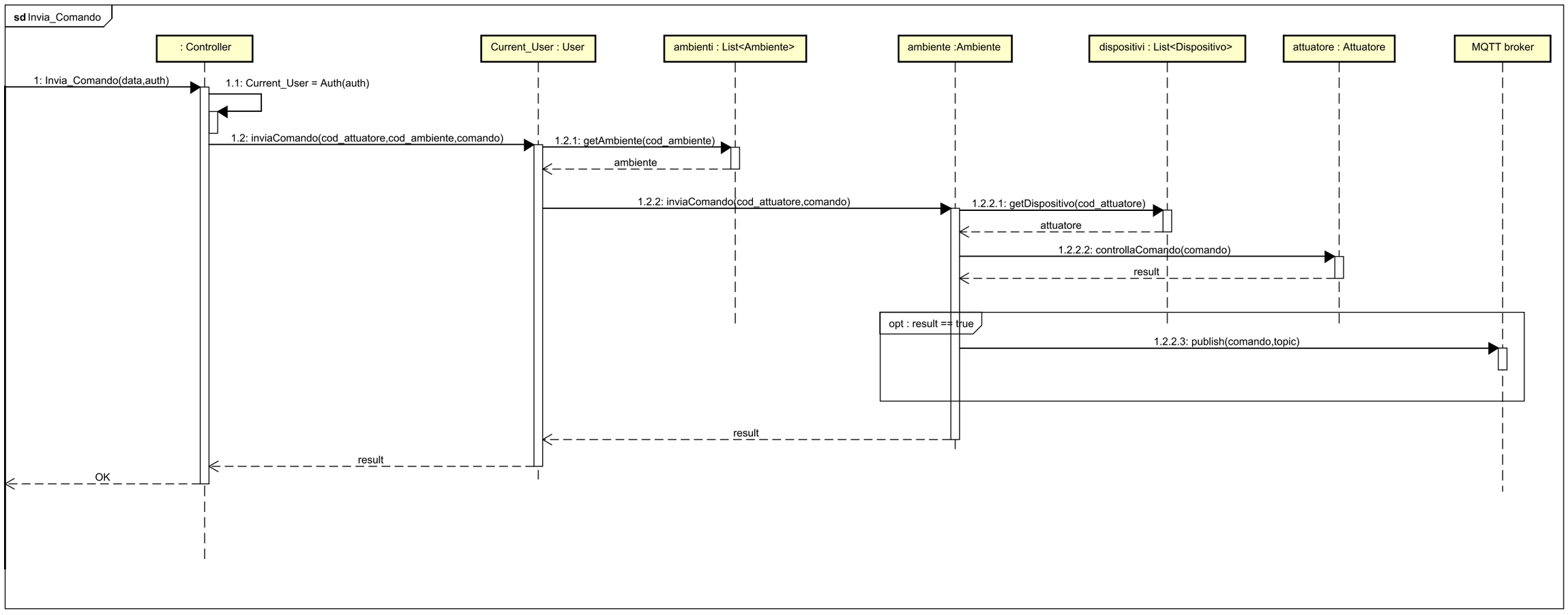
Server



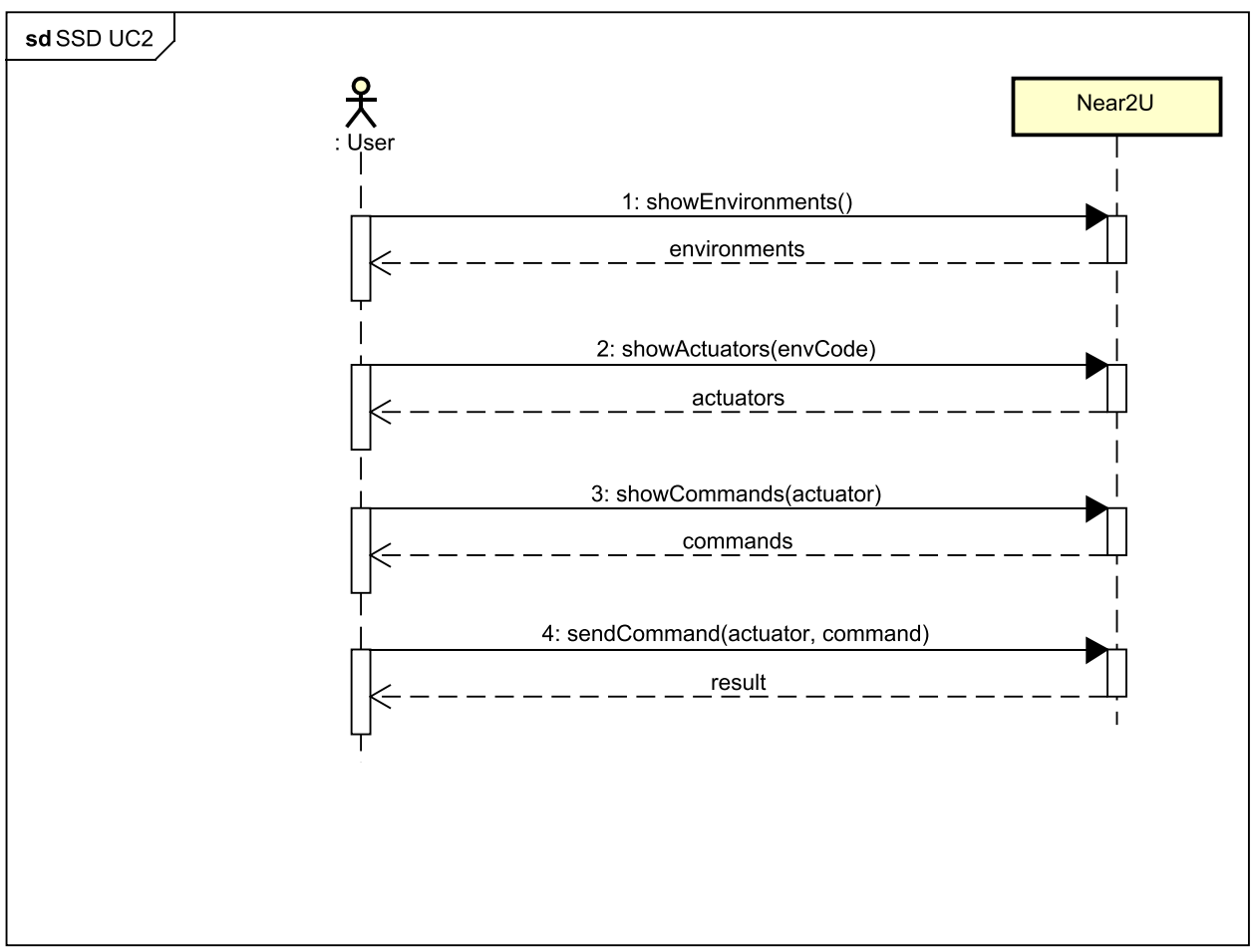
All'interno di Visualizza Dispositivi il client indicherà se vuole visualizzare sensori o attuatori

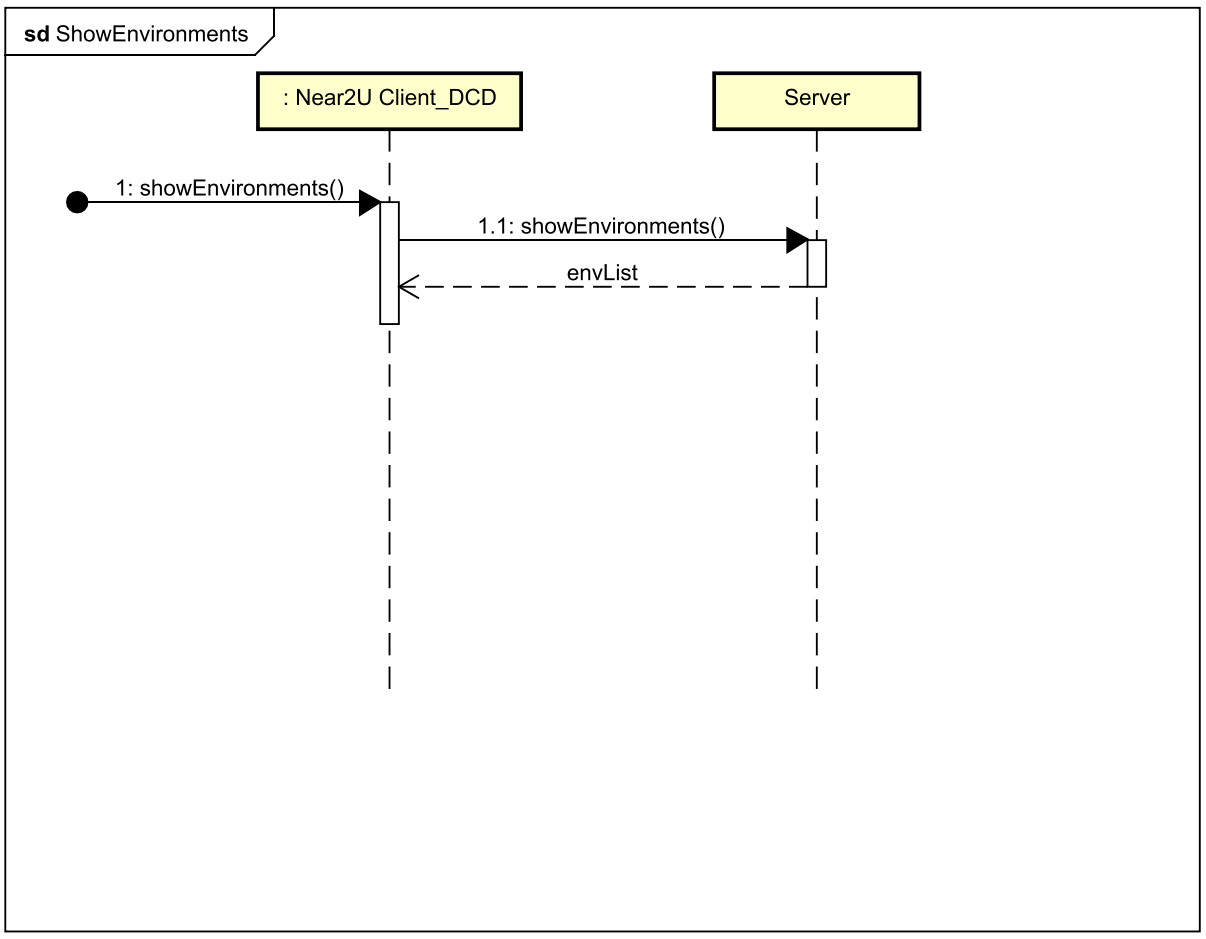

sd Visualizza_Ambienti
Controller
Current_User : User
1: Visualizza_Ambienti(data)
1.1: Current_User = Auth(auth_token)
1.2: getAmbienti()
ambienti
data

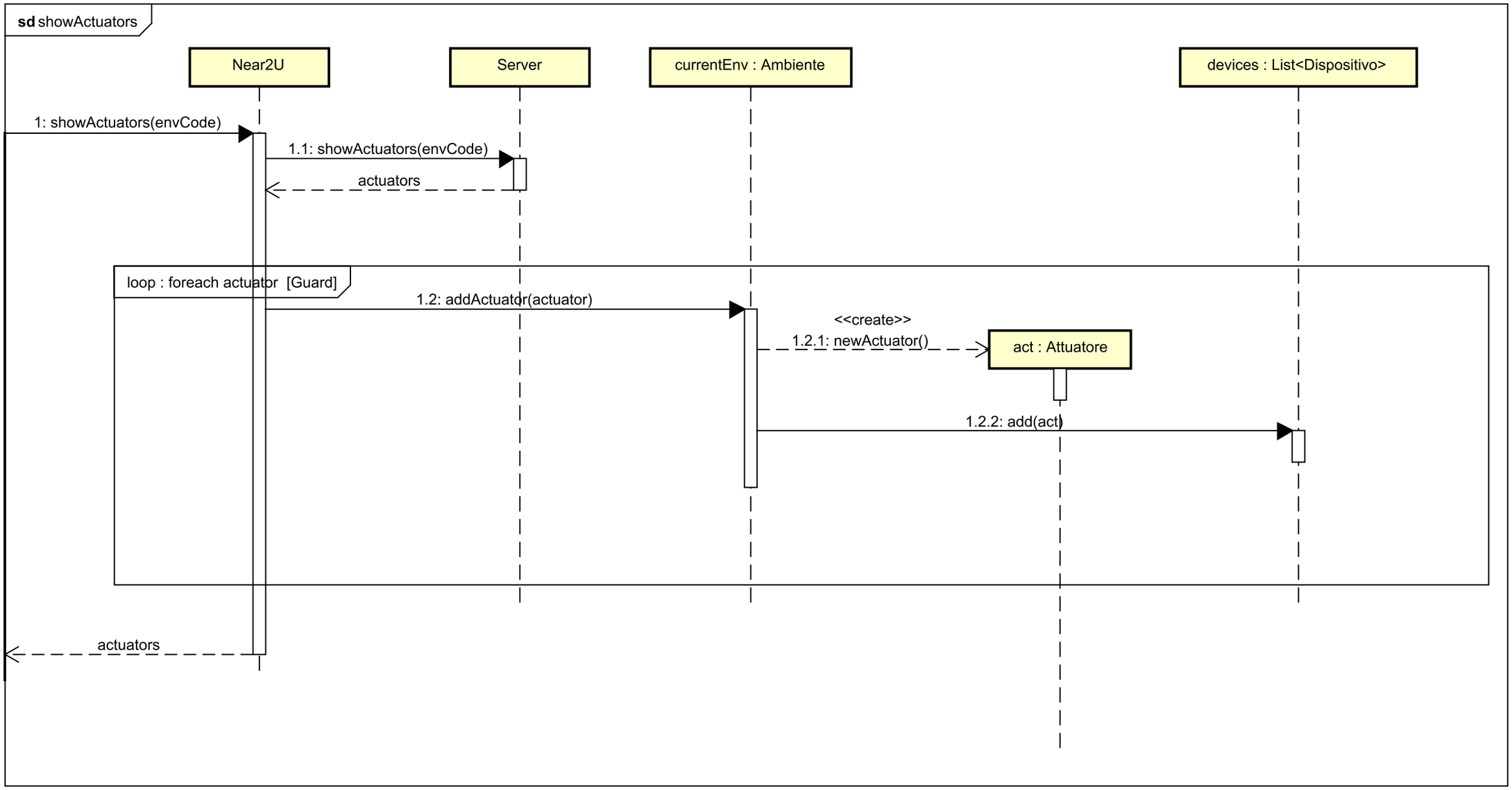

sd Visualizza_Dispositivi
Controller
Current_User : User
ambienti : List<Ambiente>
ambiente : Ambiente
1: Visualizza_Dispositivi(data)
1.1: Current_User = Auth(auth_token)
1.2: getDispositivi(cod_ambiente)
1.2.1: getAmbiente(cod_ambiente)
ambiente
1.2.2: getDispositivi()
dispositivi
dispositivi
data
Una volta ricevuta la lista di dispositivi viene inserita in un Json Array ed ogni elemento conterrà un dispositivi con tutte le informazioni

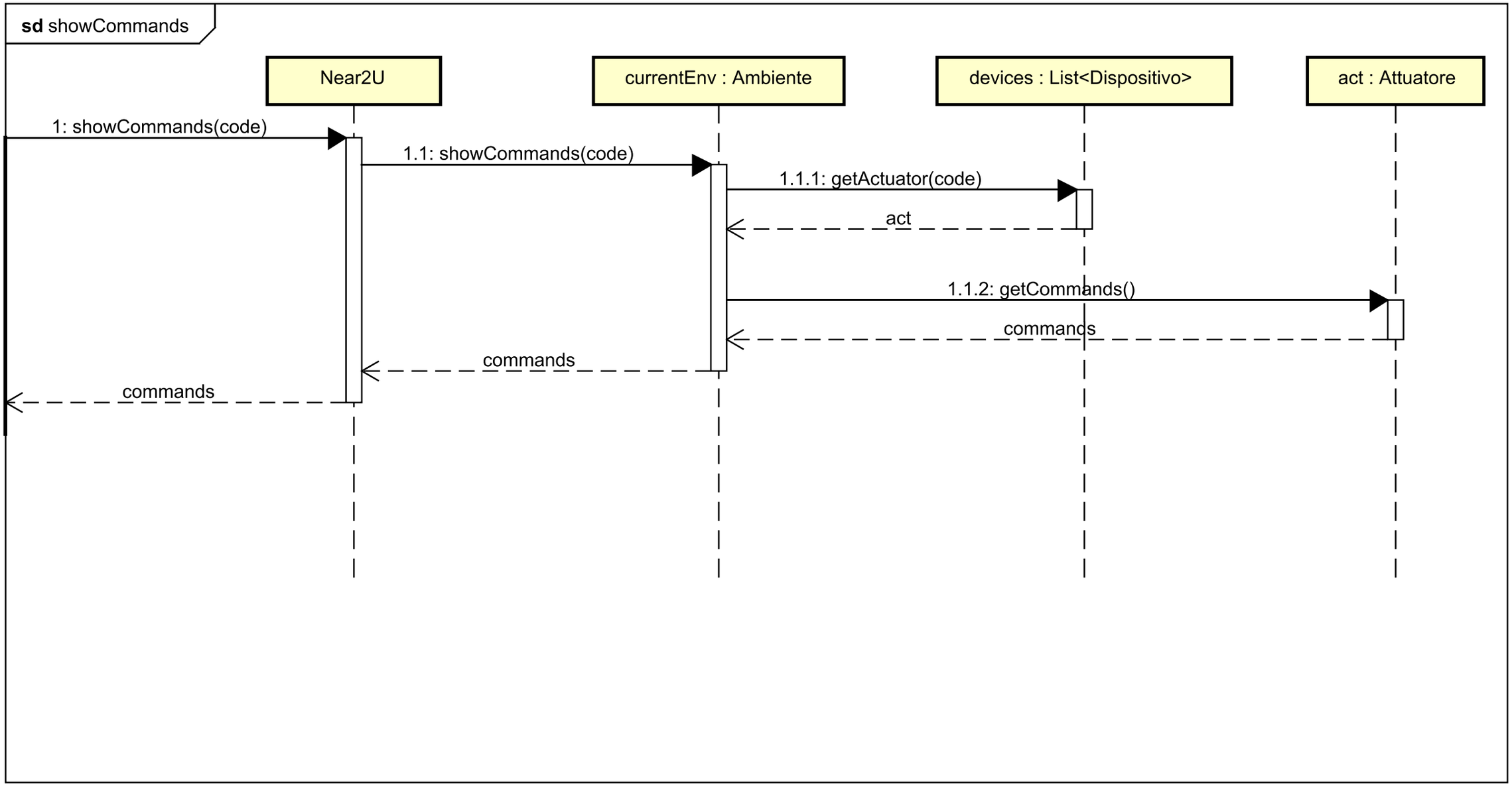

sd Invia_Comando
: Controller
Current_User : User
ambienti : List<Ambiente>
ambiente :Ambiente
dispositivi : List<Dispositivo>
attuatore : Attuatore
MQTT broker
1: Invia_Comando(data,auth)
1.1: Current_User = Auth(auth)
1.2: inviaComando(cod_attuatore,cod_ambiente,comando)
1.2.1: getAmbiente(cod_ambiente)
ambiente
1.2.2: inviaComando(cod_attuatore,comando)
1.2.2.1: getDispositivo(cod_attuatore)
attuatore
1.2.2.2: controllaComando(comando)
result
opt : result == true
1.2.2.3: publish(comando,topic)
result
result
OK

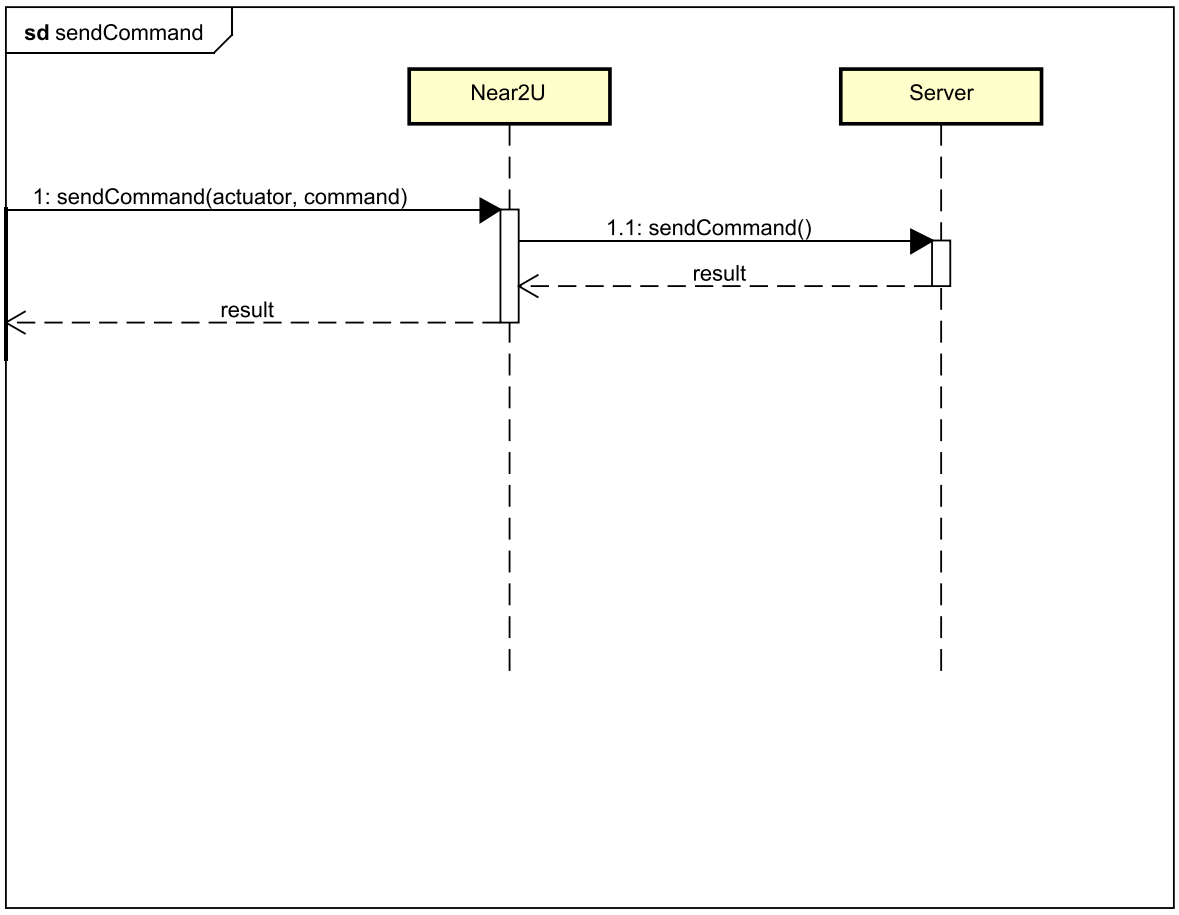
Client


sd SSD UC2
: User
Near2U
1: showEnvironments()
environments
2: showActuators(envCode)
actuators
3: showCommands(actuator)
commands
4: sendCommand(actuator, command)
result


sd ShowEnvironments
: Near2U Client_DCD
Server
1: showEnvironments()
1.1: showEnvironments()
envList


sd showActuators
Near2U
Server
currentEnv : Ambiente
devices : List<Dispositivo>
1: showActuators(envCode)
1.1: showActuators(envCode)
actuators
loop : foreach actuator [Guard]
1.2: addActuator(actuator)
<<create>>
1.2.1: newActuator()
act : Attuatore
1.2.2: add(act)
actuators

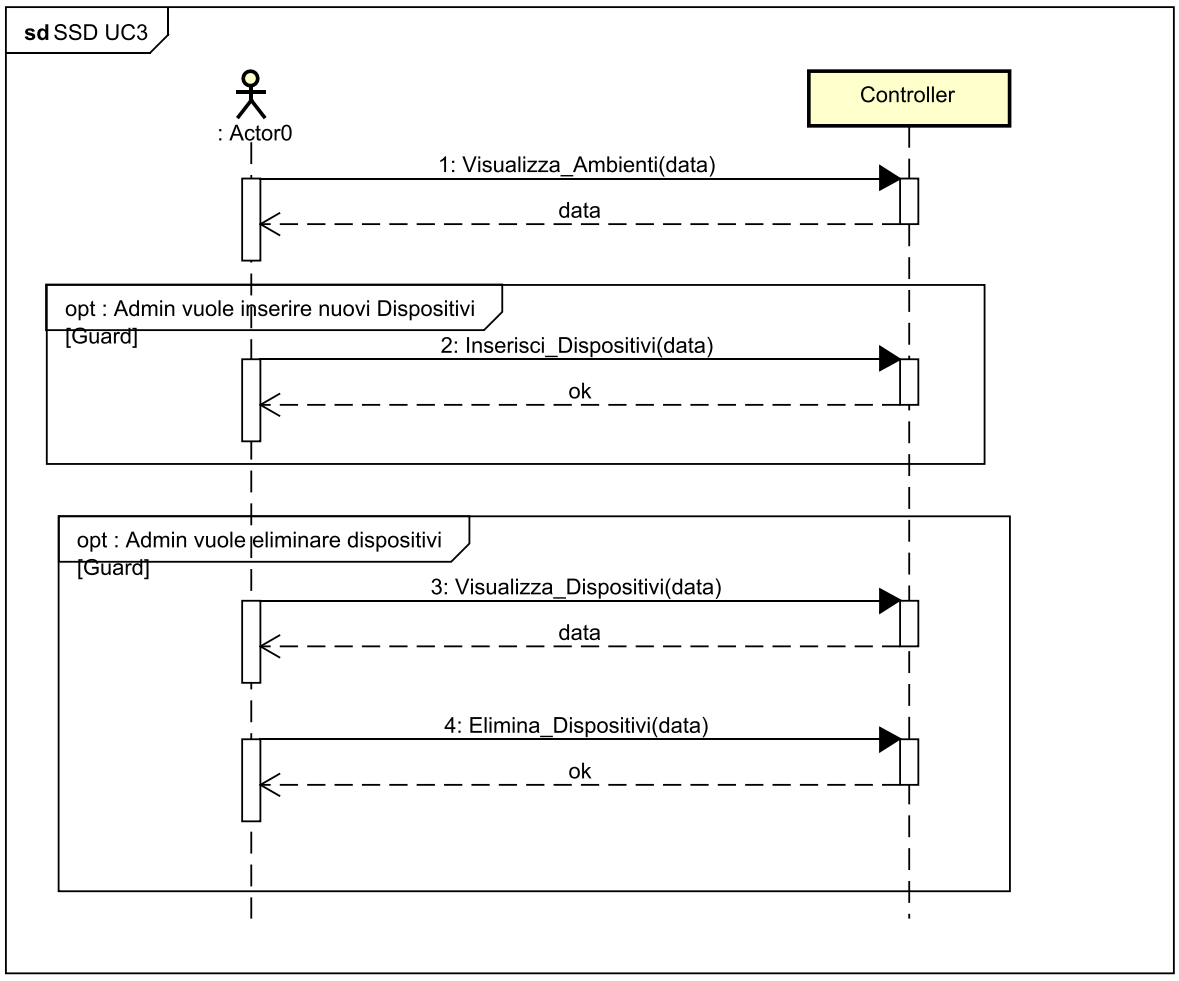

sd showCommands
Near2U
currentEnv : Ambiente
devices : List<Dispositivo>
act : Attuatore
1: showCommands(code)
1.1: showCommands(code)
1.1.1: getActuator(code)
act
1.1.2: getCommands()
commands
commands
commands

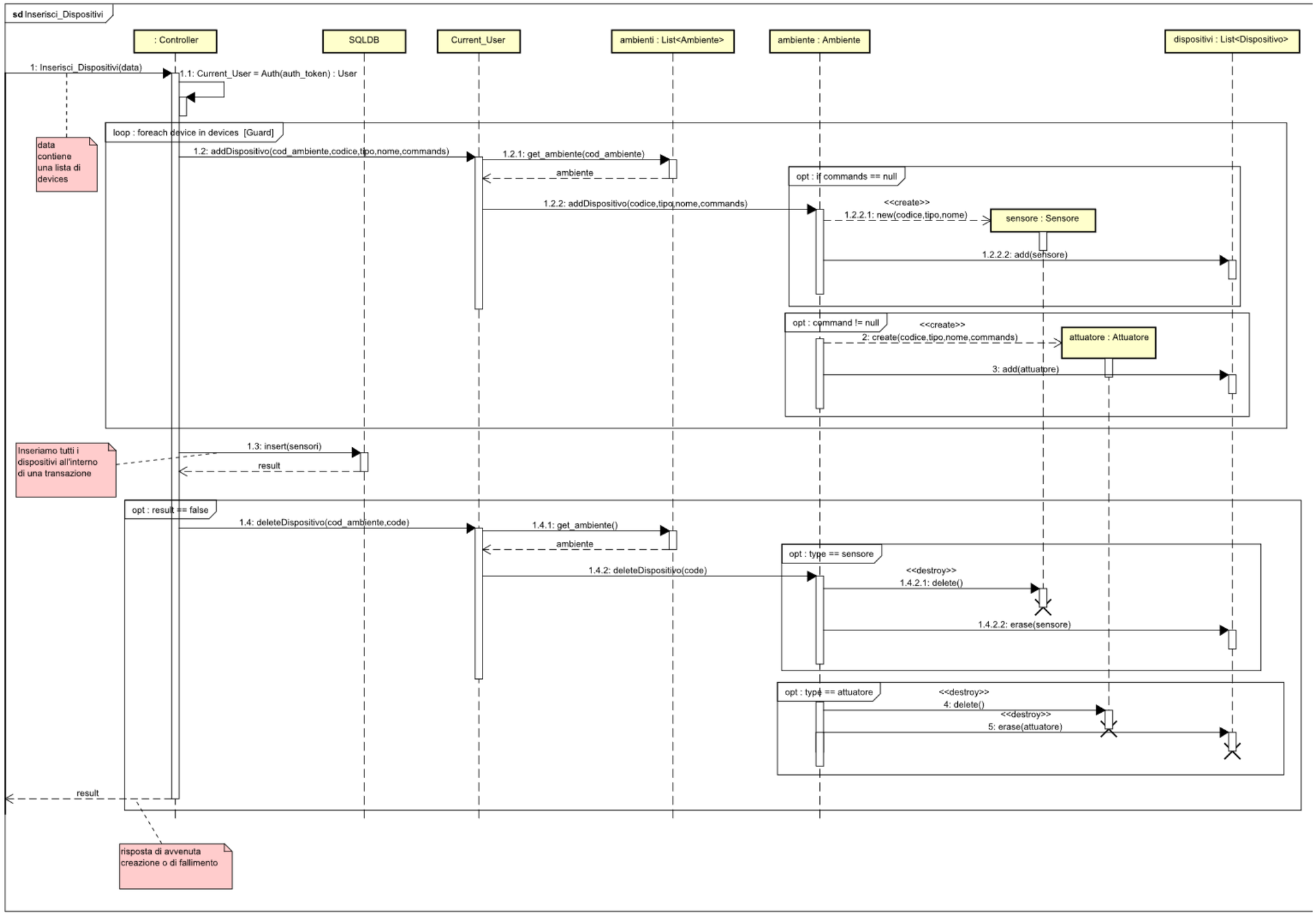

sd sendCommand
Near2U
Server
1: sendCommand(actuator, command)
1.1: sendCommand()
result
result

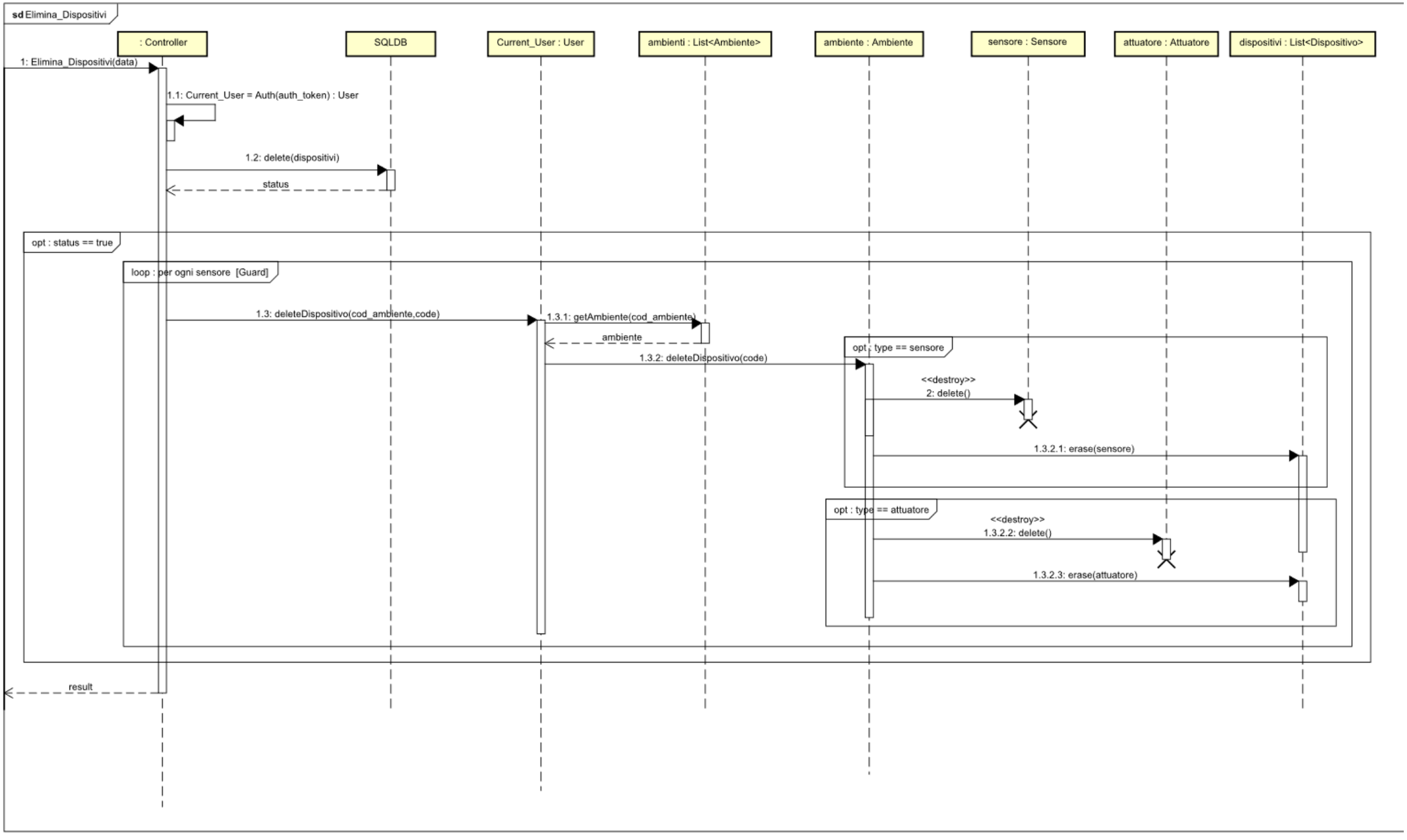
UC3: Configura ambiente

L’admin vuole configurare un nuovo ambiente o un nuovo ambiente, aggiungendo o cancellando dispositivi. Di seguito vengono proposti i diagrammi di sequenza, rispettivamente per il server e per il client inerenti al caso d’uso.
Per garantire l’unicità dell’ambiente solo all’interno degli utenti associati a quest’ultimo come chiave si è scelto il codice ambiente che sarà composto da email_admin + nome ambiente.

Server



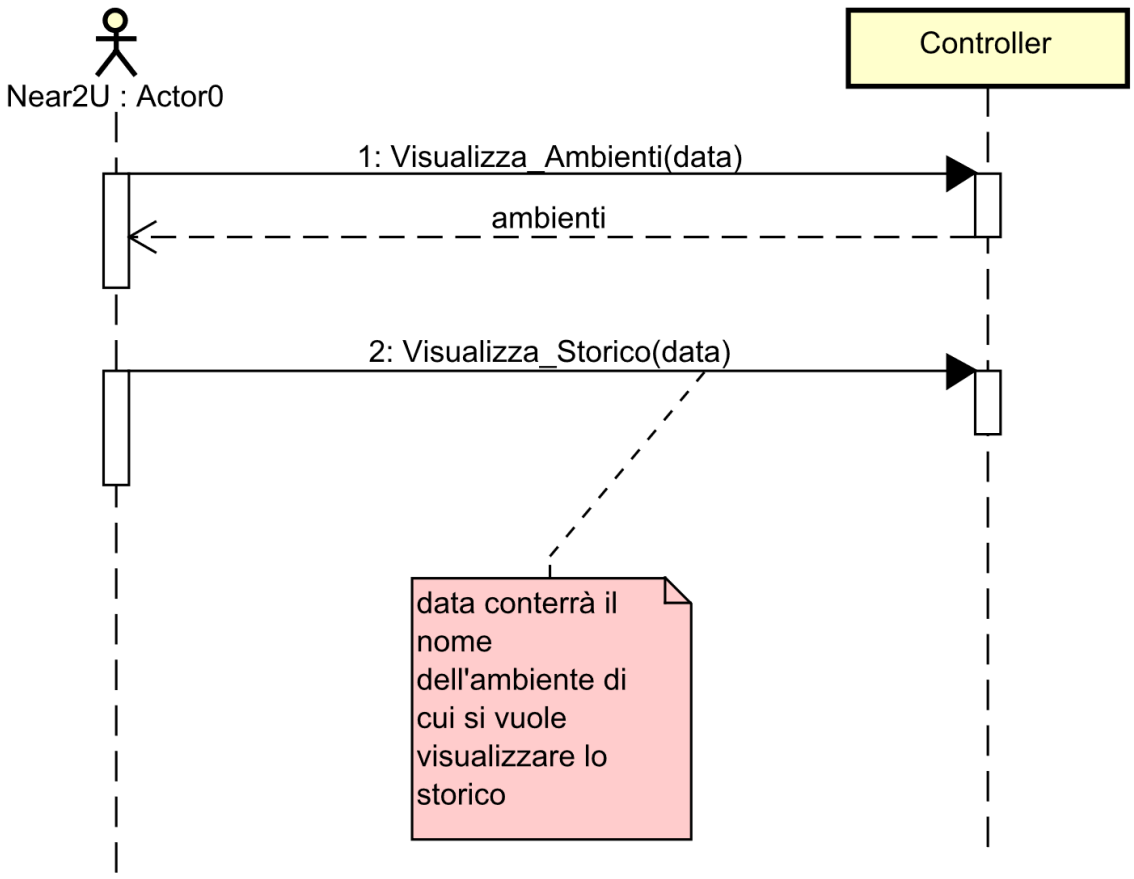

sd Inserisci_Dispositivi
: Controller
SQLDB
Current_User
ambienti : List<Ambiente>
ambiente : Ambiente
dispositivi : List<Dispositivo>
1: Inserisci_Dispositivi(data)
1.1: Current_User = Auth(auth_token) : User
loop : foreach device in devices [Guard]
data contiene una lista di devices
1.2: addDispositivo(cod_ambiente,codice,tipo,nome,commands)
1.2.1: get_ambiente(cod_ambiente)
ambiente
opt : if commands == null
1.2.2: addDispositivo(codice,tipo,nome,commands)
<<create>>
1.2.2.1: new(codice,tipo,nome)
sensore : Sensore
1.2.2.2: add(sensore)
opt : command != null
<<create>>
2: create(codice,tipo,nome,commands)
attuatore : Attuatore
3: add(attuatore)
1.3: insert(sensori)
result
Inseriamo tutti i dispositivi all'interno di una transazione
opt : result == false
1.4: deleteDispositivo(cod_ambiente,code)
1.4.1: get_ambiente()
ambiente
opt : type == sensore
1.4.2: deleteDispositivo(code)
<<destroy>>
1.4.2.1: delete()
1.4.2.2: erase(sensore)
opt : type == attuatore
<<destroy>>
4: delete()
<<destroy>>
5: erase(attuatore)
result
risposta di avvenuta creazione o di fallimento

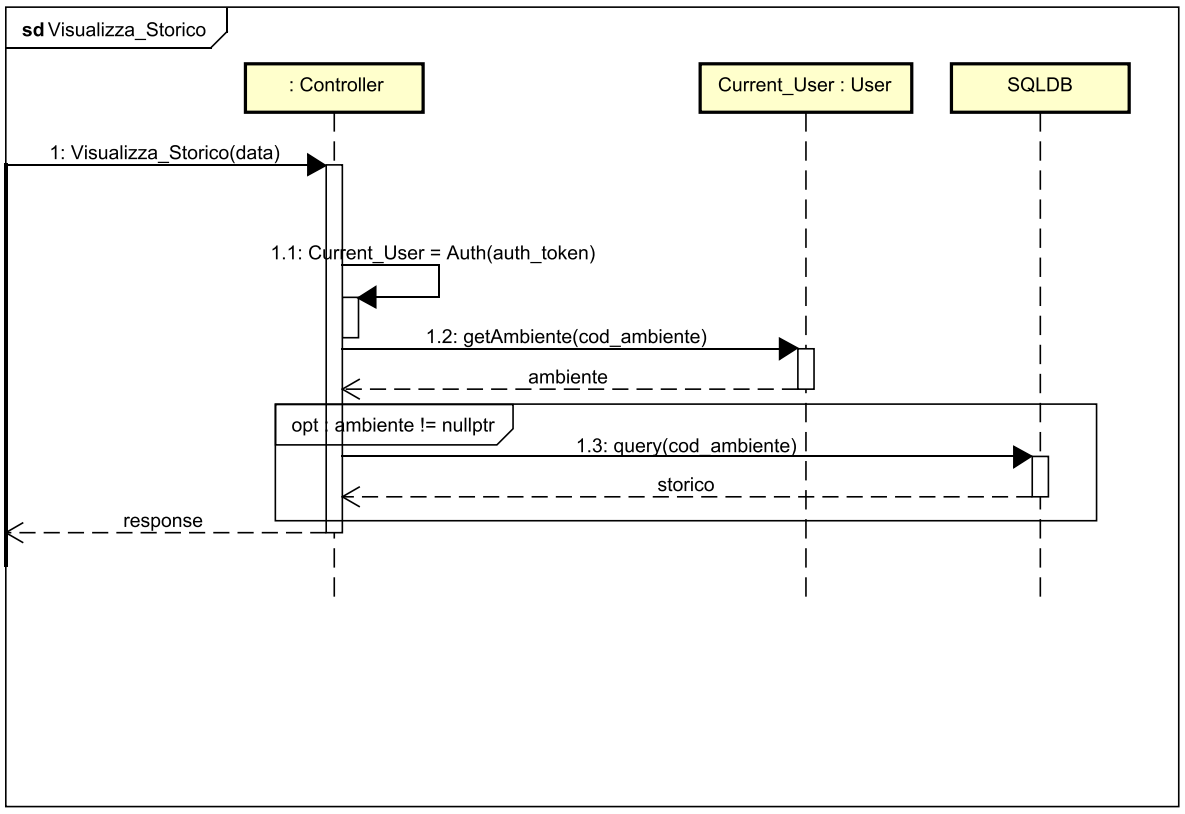

sd Elimina_Dispositivi
: Controller
SQLDB
Current_User : User
ambienti : List<Ambiente>
ambiente : Ambiente
sensore : Sensore
attuatore : Attuatore
dispositivi : List<Dispositivo>
1: Elimina_Dispositivi(data)
1.1: Current_User = Auth(auth_token) : User
1.2: delete(dispositivi)
status
opt : status == true
loop : per ogni sensore [Guard]
1.3: deleteDispositivo(cod_ambiente,code)
1.3.1: getAmbiente(cod_ambiente)
ambiente
1.3.2: deleteDispositivo(code)
opt : type == sensore
<<destroy>>
2: delete()
1.3.2.1: erase(sensore)
opt : type == attuatore
<<destroy>>
1.3.2.2: delete()
1.3.2.3: erase(attuatore)
result

UC4: Visualizza storico

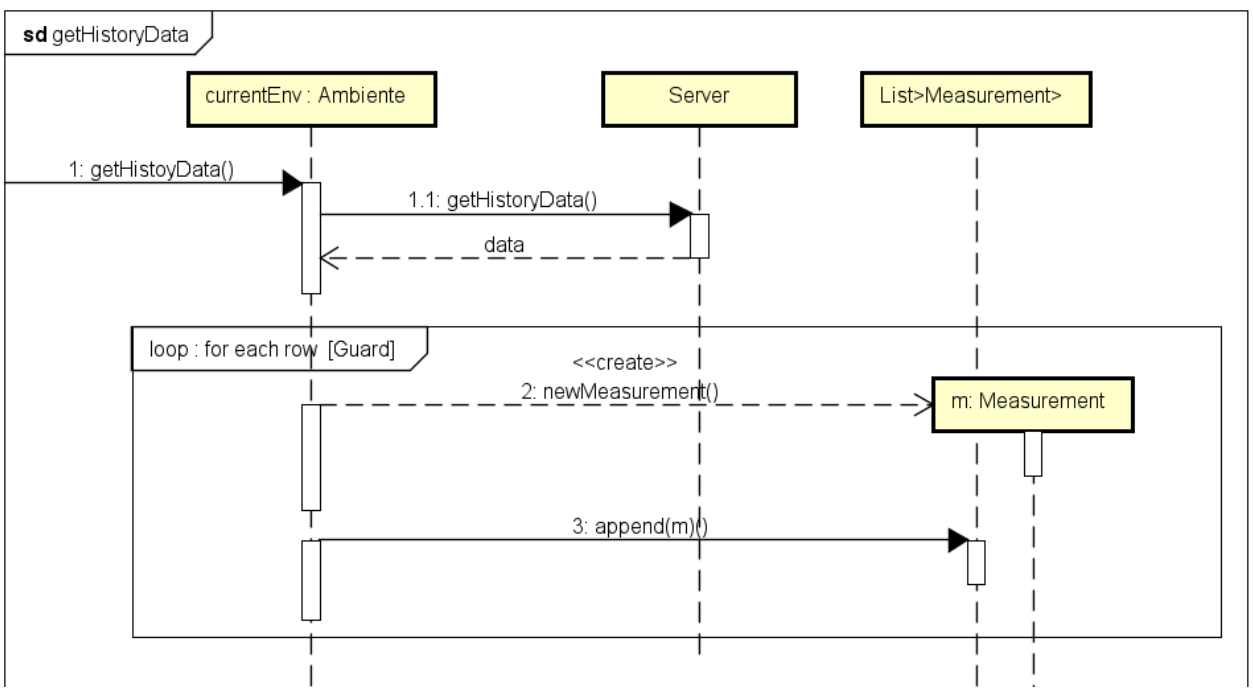
L’utente vuole visualizzare tutti i dati acquisiti dai sensori di un Ambiente. In questo caso in client invierà una richiesta al server contenente il nome dell’ambiente di cui si vuole visualizzare lo storico. Il server attraverso una richiesta al database risponderà con i dati acquisiti.
Per il prelievo dei dati dai sensori ogni qual volta viene creato un ambiente o viene effettuato il login il client Mqtt relativo al server effettua una sottoscrizione al topic dell’ambiente e per ogni dato ricevuto lo inserisce all’interno del database per garantire persistenza.
Di seguito vengono proposti i diagrammi di sequenza, rispettivamente per il server e per il client inerenti al caso d’uso.

Server




sd Visualizza_Storico
: Controller
Current_User : User
SQLDB
1: Visualizza_Storico(data)
1.1: Current_User = Auth(auth_token)
1.2: getAmbiente(cod_ambiente)
ambiente
opt : ambiente != nullptr
1.3: query(cod_ambiente)
storico
response

Client

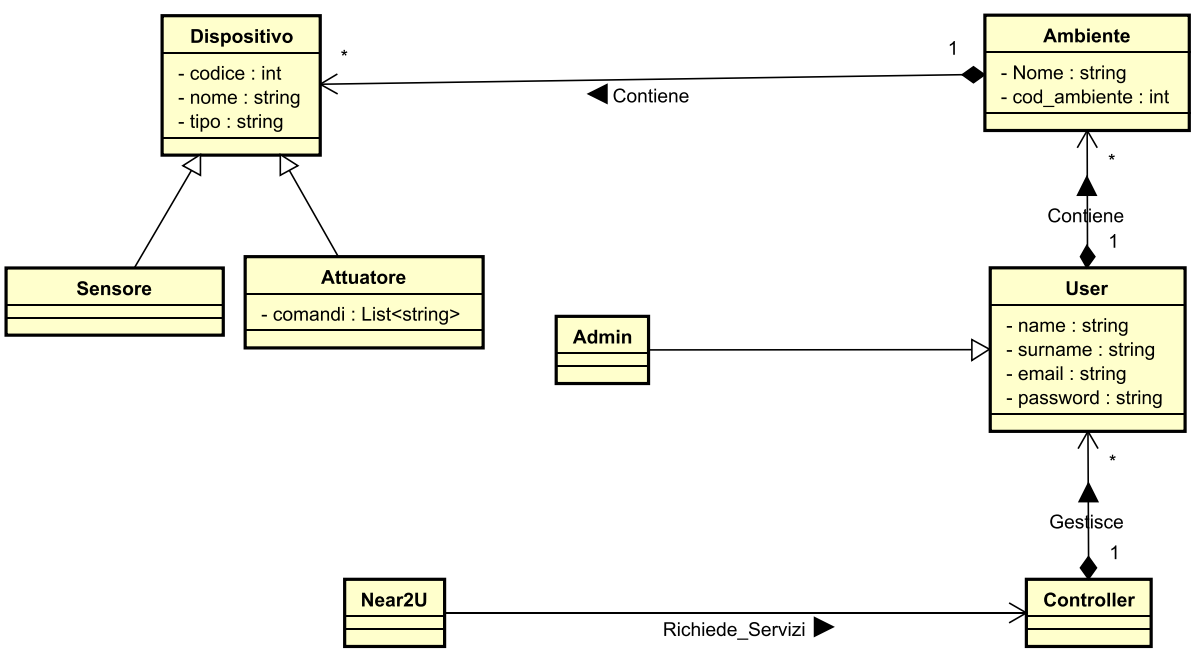

sd getHistoryData
currentEnv : Ambiente
Server
List>Measurement>
1: getHistoyData()
1.1: getHistoryData()
data
loop : for each row [Guard]
<<create>>
2: newMeasurement()
m: Measurement
3: append(m)()

Modello di Dominio

Server

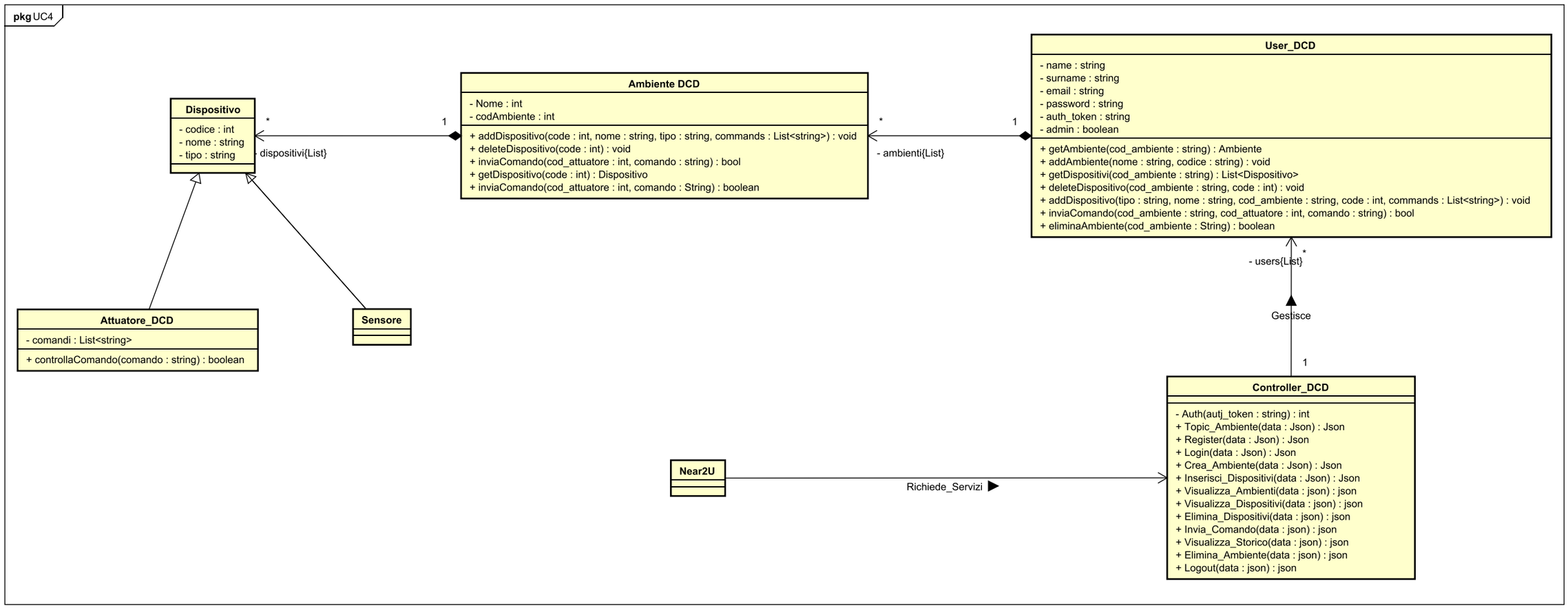
è stato possibile identificare le seguenti classi concettuali:

- **Near2U:** Rappresenta il client che invia le richieste al server
- **Controller:** Rappresenta il Server che gestirà le richieste
- **Utente:** Rappresenta il fruitore dell'app.
- **Ambiente:** Rappresenta l'ambiente in cui si vogliono effettuare operazioni
- **Dispositivo:** Rappresenta gli elementi presenti all'interno dell'ambiente
- **Sensore:** Rappresenta una tipologia di dispositivo.
- **Attuatore:** Rappresenta una tipologia di dispositivo

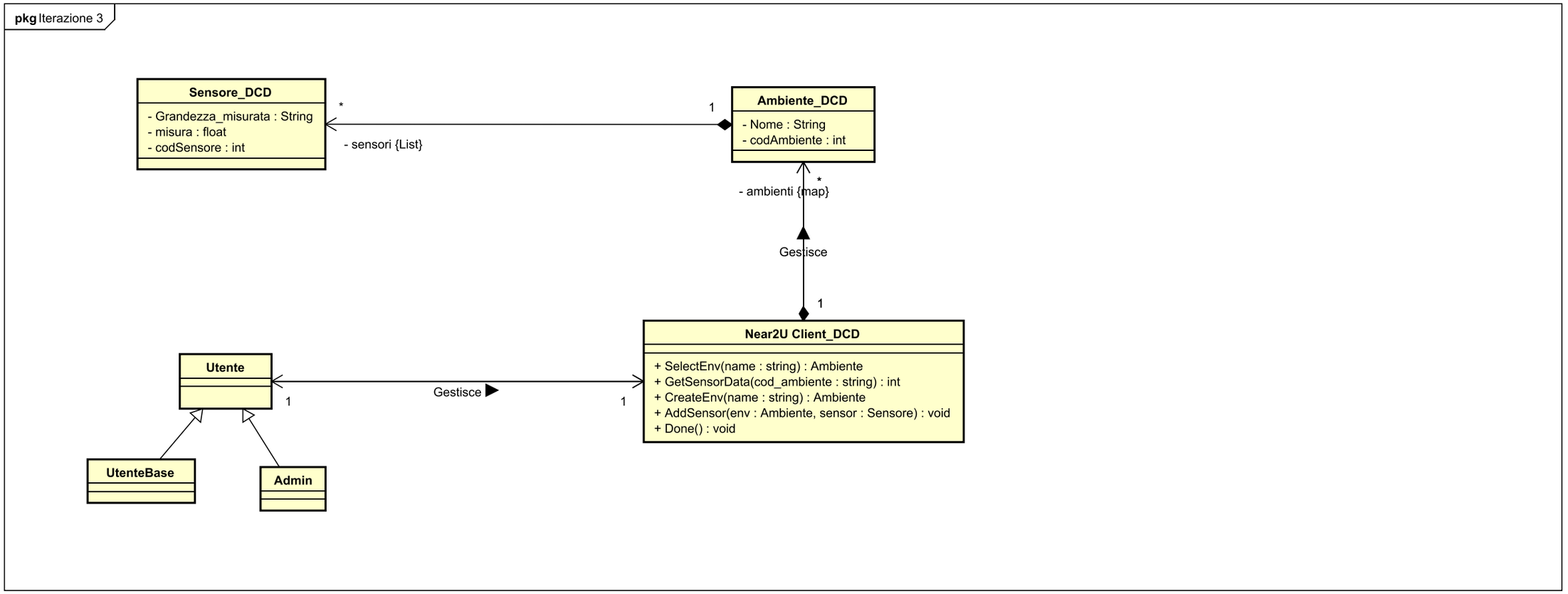


Diagrammi delle classi di progetto

Server



Client


pkg Iterazione 3
Sensore_DCD
- Grandezza_misurata : String
- misura : float
- codSensore : int
*
- sensori {List}
1
Ambiente_DCD
- Nome : String
- codAmbiente : int
*
- ambienti {map}
Gestisce
1
Near2U Client_DCD
+ SelectEnv(name : string) : Ambiente
+ GetSensorData(cod_ambiente : string) : int
+ CreateEnv(name : string) : Ambiente
+ AddSensor(env : Ambiente, sensor : Sensore) : void
+ Done() : void
Utente
1
Gestisce
1
UtenteBase
Admin

Funzionalità aggiuntive

Oltre alle funzionalità indicate nei casi d'uso sono state implementate delle funzionalità aggiuntive che sono fondamentali per il funzionamento del sistema:

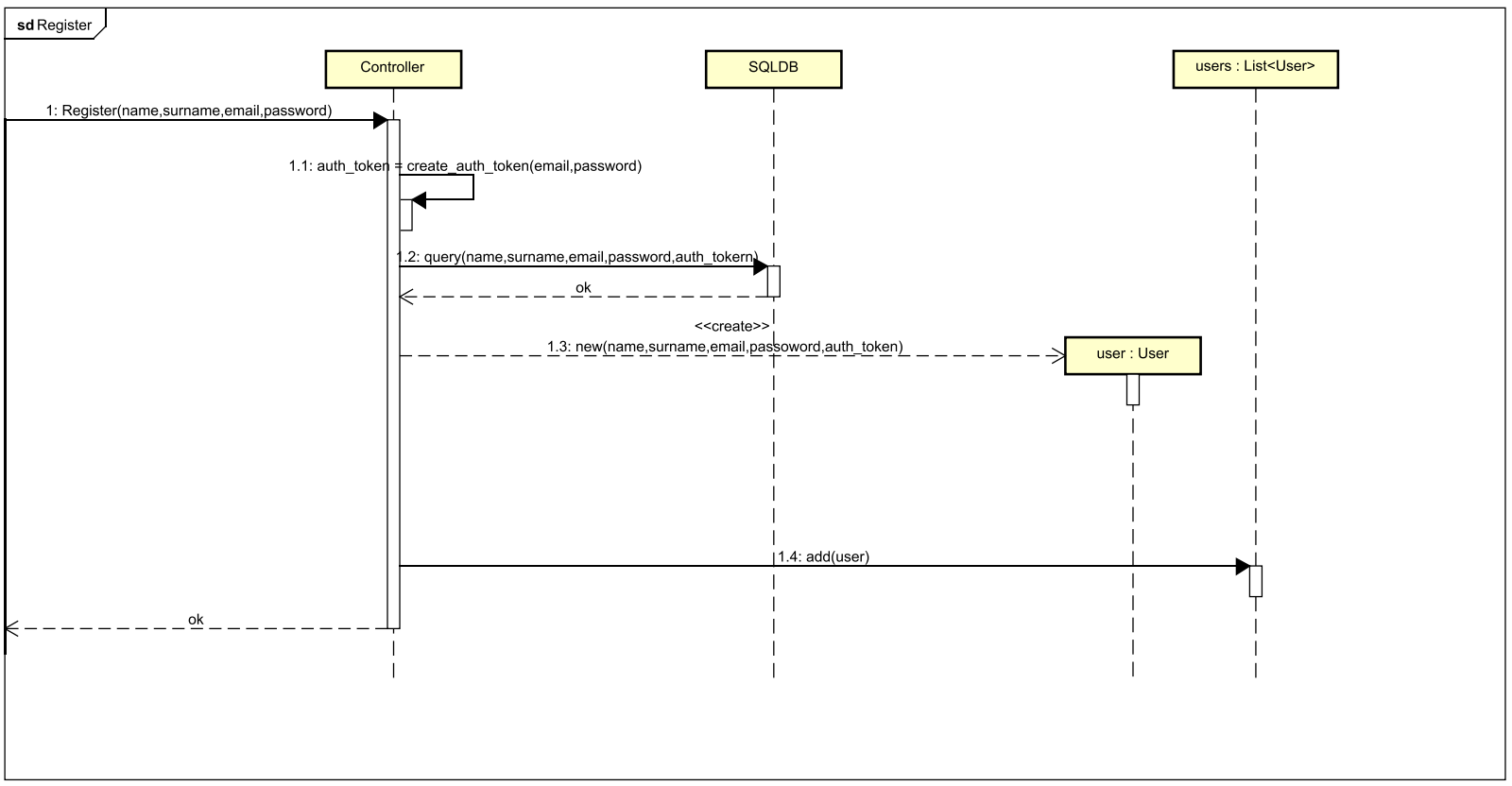
- Login
- Register
- Logout
- Elimina Ambiente
- Associa Ambiente

Register

Per la registrazione il client invierà una richiesta contenente nome,cognome,email,password, il server aggiungere al DB l'utente aggiungendo 2 campi:

- auth_token: token di autorizzazione calcolato all'interno del server eseguendo l'SHA digest di email+password
- Admin: booleano che indica se l'utente risulta admin o no. (per settare questa impostazione non è stata prevista alcuna funzionalità, va impostata dal DB)

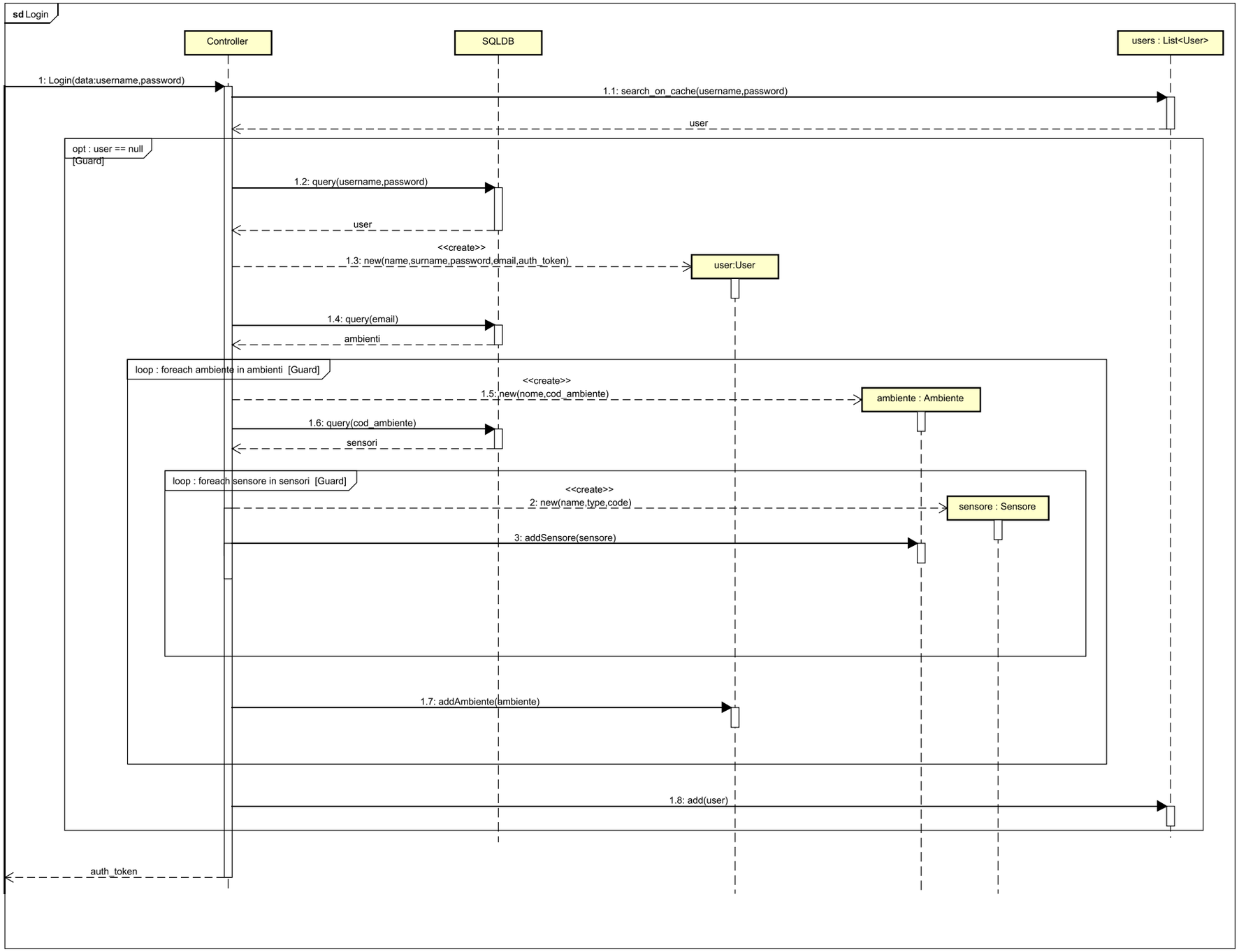
Server



Login

Nel login il client invia le credenziali e il server risponde con l'auth token che servirà ad autenticarsi in tutte le richieste successive. Il server scarica dal db tutte le informazioni inerenti l'utente (ambienti, dispositivi) di modo tale da non dover più ricontattare il DB.

Server


sd Login
Controller
SQLDB
users : List<User>
1: Login(data:username,password)
1.1: search_on_cache(username,password)
user
opt : user == null
[Guard]
1.2: query(username,password)
user
<<create>>
1.3: new(name,surname,password,email,auth_token)
user:User
1.4: query(email)
ambienti
loop : foreach ambiente in ambienti [Guard]
<<create>>
1.5: new(nome,cod_ambiente)
ambiente : Ambiente
1.6: query(cod_ambiente)
sensori
loop : foreach sensore in sensori [Guard]
<<create>>
2: new(name,type,code)
sensore : Sensore
3: addSensore(sensore)
1.7: addAmbiente(ambiente)
1.8: add(user)
auth_token

Logout

Con questa richiesta l’utente si disconnette dall’app e tutti i dati memorizzati in locale nel server vengono eliminati.

Elimina Ambiente

L’admin invia una richiesta contenente il codice dell’ambiente e il server elimina sia in locale che nel DB tutte le informazioni su quell’ambiente.

NB Si sono inseriti solo i grafici relative ad operazioni più complesse.