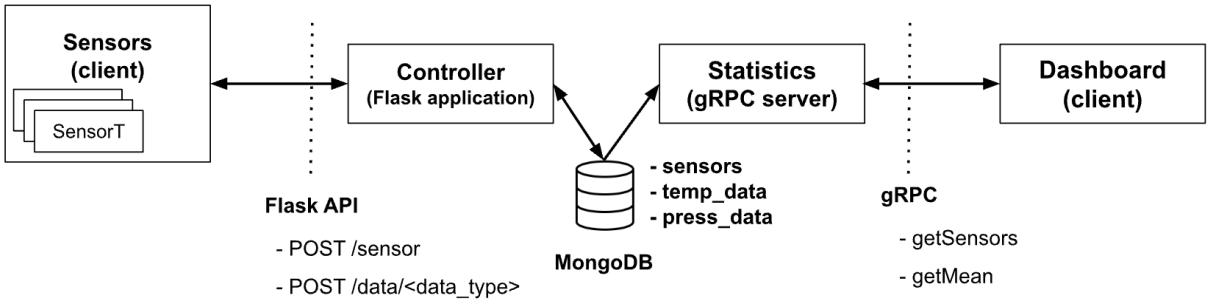
Università degli Studi di Napoli Federico II
Advanced Computer Programming
Esercitazione 05
gRPC, Flask e MongoDB

Il candidato realizzi un'applicazione Python basata su **gRPC**, **Flask** e **MongoDB** per la gestione di **sensori remoti**. Ogni sensore è caratterizzato da un ***_id*** (intero), che lo identifica univocamente, e da un ***data_type*** (stringa), che indica se si tratta di un sensore di *temperatura* (temp) o di *pressione* (press).
Le misurazioni effettuate da un sensore, invece, sono caratterizzate dai campi ***sensor_id*** (intero), cioè l'id del sensore, e ***data*** (intero), che contiene il valore della misurazione.
Il sistema è composto da 4 entità, come illustrato in figura.



Controller è una Flask web application che implementa una REST API con le seguenti Route HTTP:

- **POST /sensor**: permette di registrare un sensore presso il Controller. Il Controller riceve le informazioni del sensore (in formato JSON) attraverso il body della richiesta POST, e.g., `{"_id":10, "data_type":"temp"}`. Tali informazioni sono inserite come document in una collection MongoDB denominata *sensors*. Il Controller restituisce una risposta (in formato JSON) contenente l'esito dell'operazione, e.g., `{"result": "success"}`. Da notare che l'operazione potrebbe fallire nel caso in cui il sensore sia già registrato.
- **POST /data/<data_type>:** permette di inviare la misurazione effettuata dal sensore al Controller. Il Controller riceve la misurazione (in formato JSON) attraverso il body della richiesta, e.g., `{"sensor_id":10, "data":30}`. Mentre la tipologia di misurazione è fornita attraverso l'URL, i.e., `<data_type>`. Ricevuta la richiesta il Controller, salva la misurazione come document nella collection *temp_data* se data_type è pari a temp, mentre salva la misurazione nella collection *press_data* se data_type è pari a press. Il Controller restituisce una risposta (in formato JSON) contenente l'esito dell'operazione, e.g., `{"result": "success"}`. Da notare che l'operazione potrebbe fallire nel caso in cui venga richiesto un data_type non previsto.

Sensors: simula un set di sensori che interagiscono con il Controller. All'avvio sono generati 5 thread, ognuno dei quali rappresenta un sensore. Ad ogni thread è assegnato un *id* incrementale (utilizzato come *_id* del sensore) ed un *data_type* scelto a caso tra *temp* e *press*. Ogni thread all'avvio effettua la registrazione presso il Controller attraverso */sensor*. Successivamente, effettua 5 misurazioni invocando */data/<data_type>*. Il campo *data* della misurazione è scelto in maniera casuale tra 1 e 50.

Statistics: server gRPC che offre i servizi descritti dal file `statistics.proto` fornito:

- **rpc getSensors(Empty) returns (stream Sensor)**: permette di recuperare dalla collection *sensors* i dati dei sensori registrati presso il Controller. Dopo aver recuperato i dati, la funzione ritorna uno stream con le informazioni dei singoli sensori (*Sensor*).
- **rpc getMean(MeanRequest) returns (StringMessage)**: permette di calcolare la media delle misurazioni fatte da un determinato sensore. La funzione riceve le informazioni del sensore, cioè *sensor_id* e *data_type* attraverso *MeanRequest*. In base al data_type, la funzione recupera dal database le misurazioni del sensore (i.e., dalla collection *temp_data* se data_type è *temp*, e da quella *press_temp* se data_type è *press*) sfruttando il sensor_id. Successivamente, effettua il calcolo delle media delle misurazioni, il cui risultato è ritornato attraverso uno *StringMessage*.

Dashboard: client gRPC che utilizza i servizi messi a disposizione da Statistics. Precisamente la Dashboard all'avvio richiede le informazioni sui sensori invocando la *getSensors*. Successivamente, per ogni sensore ricevuto, invoca la *getMean* per richiedere la media delle rispettive misurazioni.