Relazione progetto Industrial Informatics

Studenti:

- Claudio Curto
- Manlio Puglisi

Sommario

Introduzione

Si vuole realizzare un applicativo web per il monitoraggio di alcuni dati meteo prodotti da un server OPC-UA in esecuzione su cloud Azure.

Per l'estrazione di questi dati si è deciso di utilizzare API messe a disposizione dal sito openweathermap.org.

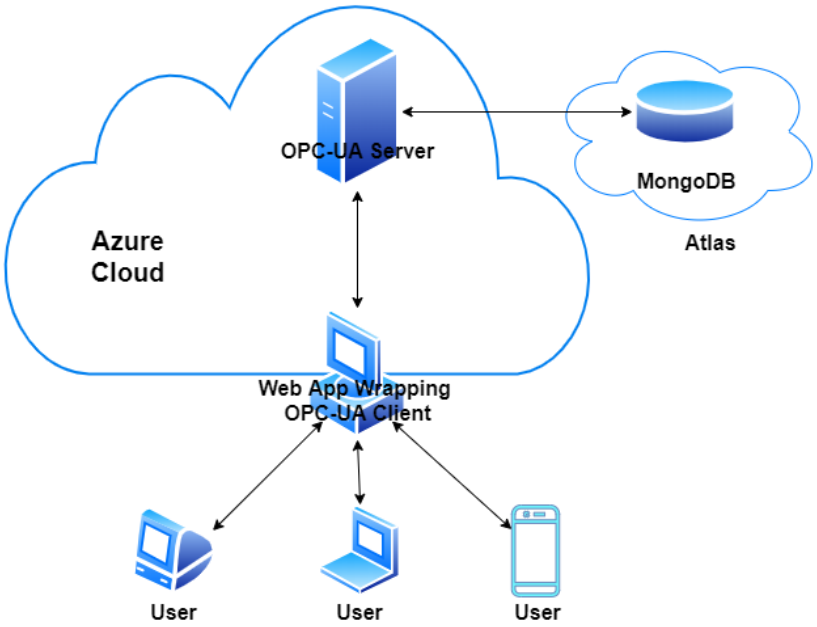
In particolare si è tenuto conto di:

- Temperatura
- Umidità
- Pressione
- Condizione meteorologica

Tecnologie utilizzate

- Node.js
- Microsoft Azure
- OPC-UA

Architettura



Descrizione

L'applicativo è costituito da tre macro componenti di cui:

- Server OPC-UA, scritto in node.js in esecuzione su cloud Azure
- Client OPC-UA, scritto anch'esso in node.js, racchiuso in una web application express.js
- Database MongoDB

Il server è racchiuso all'interno di un container Azure. Nello stesso container, è racchiuso un server scritto in Node.js che tramite Express.js si occupa di tre funzioni: la prima è esporre una landing page in cui troviamo un form per inserire delle credenziali; la seconda, al click del tasto *Sign In*, si occupa di inoltrare le credenziali inserite nel form al client OPC-UA che tenterà di stabilire una connessione con server OPC-UA in

esecuzione nello stesso container; la terza, nel caso in cui le credenziali siano corrette, esporre una view in formato *.ejs* contenente una tabella in cui vengono caricate le informazioni ottenute dal client.

Server-OPCUA

Il compito del server è di produrre feed meteo tramite chiamate API alla piattaforma openweather.org. Le chiamate API sono state implementate in maniera asincrona nel file utility.js, successivamente importato nel file applicativo principale (secure_server.js).

```
let getCityWeather = async (city) => {
    let result = await new Promise((resolve) => {
        unirest.get(
            `http://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${key}`
            )
            .end(
                (response) => resolve(response)
            );
    });
    if (result.status !== 200) {
        throw new Error("API error");
    }
    return result.body;
}
```

Una volta effettuata la chiamata si è proceduto al parsing dei valori di nostro interesse tramite la funzione "extractUsefulData".

Per ogni città si è deciso di definire un ObjectNode con reference "OrganizedBy" rispetto ad un Object "Cities" e si è costruito per ognuno di questi il proprio set di variabili. Il programma è stato realizzato in un'ottica scalabile in base al numero di città di interesse, andando a leggere tali città da un array definito in un file javascript situato nella cartella **utility**, modificabile dai programmatori. Seguendo l'idea dell'apporto di migliorie e aggiunte future, si potrebbe pensare anche di definire un file testuale che viene letto a runtime per permettere ad eventuali utenti di aggiungere e/o rimuovere città in base alle loro esigenze, senza toccare una linea di codice.

Durante la realizzazione del Server si è posta particolare attenzione sull'instaurazione di una connessione sicura col client, in particolare sono stati definiti i parametri "userManager", per la gestione del login da parte del client tramite username e password, e "certificateManager" per quanto riguarda i certificati di sicurezza:

```
let userManager = {
  isValidUserAsync: (userName, password, callback) => {
    query.findUser(userName, password)
    .then((user) => {
      callback(null, user);
    })
```

Dal blocco di codice soprastante, si evince come venga effettuata una query per controllare se l'username e la password inserite siano presenti nel database Atlas MongoDB remoto. La funzione *findUser* è definita nella cartella **autenticazione**, in cui è contenuto pure il file **db.js** che contiene le credenziali per permettere la connessione col database.

Se le credenziali inserite sono contenute nel database, viene dato il permesso di accedere al server OPC-UA e monitorare i dati da esso fornito.

In particolare, tramite il campo *allowAnonymous: false* all'interno dei parametri di connessione, è esplicitamente indicato che le connessioni anonime - ossia senza username e password - non sono consentite.

```
const certificateManager = new opcua.OPCUACertificateManager({
  automaticallyAcceptUnknownCertificate: true,
  rootFolder: path.join("./", "certificate"),
});
```

In quest'ultimo blocco, invece, viene specificato un ulteriore parametro per la gestione dei certificati da parte del server. Il parametro *automaticallyAcceptUnknownCertificate* posto a true sta ad indicare che il server accetterà automaticamente i certificati forniti dai client OPC-UA.

Node.js, Express.js e Client OPC-UA

Express è un framework per applicazioni web Node.js che fornisce funzionalità per lo sviluppo di applicazioni web, in particolare server.

Il server Express sviluppato utilizza metodi RESTful quali GET e POST.

Il codice è visualizzabile nel file **index.js**:

```
const endpointUrl = "opc.tcp://localhost:5000/UA/IndustrialInformaticsServer";
const securityMode = coerceMessageSecurityMode(1);

const connectionOption = {
    securityMode,
    endpoint_must_exist: false,
    keepSessionAlive: true,
}
```

In questo primo blocco di codice vengono definiti alcuni parametri come:

- l'endpointUrl, ossia la porta del container in cui troviamo in ascolto il server OPC-UA: si ricorda che, dato che questa webapp Express è in esecuzione sullo stesso container del server OPC-UA, per il client OPC-UA l'host dell'endpoint sarà composto da localhost + porta del server OPC-UA;
- la securityMode settata col valore **1** indica l'utilizzo di autorizzazione con coppia Username e Password;
- *keepSessionAlive: true* si occuperà di tenere viva la sessione per un minuto (valore di default). Durante questo minuto è possibile aggiornare la pagina html per ottenere eventuali nuovi valori.

```
app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname + '/index.html'));
});
```

In questa prima GET, ciò che viene restituito dal __server Express__ è la landing page, definita nella stessa directory come ```index.html```.

```
app.post('/', (req, res) => {

    let userIdentity = {
        userName: req.body.username,
        password: req.body.password
    };
[...]
    client.createSession(userIdentity, async (err, session) => {...}
```

Questa POST contiene i vari metodi che ci portano dal creare una sessione col server fino al rendering delle informazioni ottenute dal server OPC-UA all'interno della tabella.

La userIdentity verrà recuperata dai campi del form che scatenano la post; questi verranno passati al metodo del client OPC-UA che si occuperà di creare la sessione col server OPC-UA.

Dopo aver effettuato i suoi controlli di autenticazione, il server OPC-UA, in caso affermativo, permetterà al client di effettuare la browse per ogni città e di leggere i valori delle variabili di ciascun ObjectNode.

Una volta fatto questo, le informazioni verranno raccolte in un array, convertite in formato JSON e inoltrate al file **client.ejs** che effettuerà il render di queste informazioni tramite il **view engine ejs**, importato da Express.

Il file **client.ejs** verrà quindi inoltrato all'utente, che vedrà i dati raccolti in una tabella.

L'immagine sottostante rappresenta il workflow tipico della webapp.