Relazione di progetto di "Smart City e Tecnologie Mobili"

Titolo del progetto

Numero del gruppo:

Componenti del gruppo:

Indice

1 Introduzione

Esporre l'obiettivo del progetto dandone una visione complessiva.

Devono essere illustrate le caratteristiche salienti del progetto; deve essere chiara la distinzione tra le tecnologie usate/assemblate durante lo svolgimento dell'elaborato e il contributo tecnologico/scientifico effettivamente apportato dal gruppo.

Vincoli circa la lunghezza della sezione (escluse didascalie, tabelle, testo nelle immagini, schemi):
Il progetto ha come obiettivo permettere all'utente di organizzare la propria rete di raspberry per l'accensione delle luci e per la videosorveglianza della casa, utilizzando un sito internet per la gestione da remoto.
Smarthome è stato realizzato con l'intenzione di essere cross-platform, in modo che l'utente possa utilizzare qualunque device in grado di navigare su internet per accedere ai servizi forniti, in qualunque momento desiderato.

1.1 Servizi

- Sign in e Sign Up: l'utente deve prima registrarsi al sito internet per poter usufruire del contenuto, impostando nome utente, e-mail e password; quest'ultima correttamente suttoposta ad un algoritmo di *hashing* con l'aggiunta di un valore di *salt*.

- Aggiunta di device: una volta registratosi, è necessario salvare i device che possiede l'utente, specificando il nome (unico) e il tipo di azione. Le azioni corrispondono alla funzione che il raspberry deve svolgere, ovvero accensione delle luci oppure videosorveglianza.

- Gestione dei device: una volta che l'utente ha registrato il proprio device, è necessario che il raspberry apra una comunicazione con il sito. Una volta collegato ad internet, sarà proprio il raspberry che visualizzerà su console un modulo con i dettagli da completare.

Una volta stabilita la comunicazione, sarà possibile accendere o spegnere il device, eseguire l'azione prestabilita.

- Funzionalità: accensione in real-time delle luci o videosorveglianza. In tempo reale, è possibile accendere o spegnere le luci anche da remoto, mentre la videosorveglianza utilizza un modulo di computer vision per capire se c'è stato movimento, registrando un video in cui si può osservare chi è passato nella stanza.

1.2 Tecnologie e Hardware

- NodeJs, per la comunicazione client e server, tutto scritto in Javascript. NodeJs permette anche, tramite npm, l'installazione e l'utilizzo di pacchetti specifici per affrontare varie questioni (come ad esempio il cambiamento di formato da *raw H264* a *mp4* dei video)

- MongoDB, per il database. MongoDb permette di salvare nel database molti tipi di dati utilizzando uno *Schema*, con varie modalità di *fetching* e *saving* dei dati

- Raspberry PI 3, Model B. Il Raspberry è un dispositivo altamente flessibile e grazie alla sua possibilità di interfacciarsi con circuito elettrico e camera, è stata la scelta più ovvia e naturale per creare un ambiente smart-home.

- Raspberry Pi Camera V2. Questo modulo è stato interfacciato con il raspberry e viene controllato per fare computer vision

2 Stato dell'arte

Riassumere le soluzioni presenti in letteratura inerenti al problema in esame. Per ciascuna, discutere le principali diversità o affinità rispetto al progetto presentato. Nel caso non siano presenti soluzioni direttamente comparabili a quella presentata descrivere comunque le principali tecniche note per affrontare la tematica trattata.

Le soluzioni esposte devono essere corredate degli opportuni riferimenti bibliografici. Nel caso si tratti di soluzioni già operative sul mercato, devono essere indicate le fonti (online) dove poter accedere al servizio o approfondirne i contenuti.

Vincoli circa la lunghezza della sezione (escluse didascalie, tabelle, testo nelle immagini, schemi):

	Numero minimo di battute	Numero massimo di battute
1 componente	2000	3000
2 componenti	2500	4500
3 componenti	3000	6000

Il primo problema da risolvere è stato come far interagire sito internet e raspberry, ovvero trovare una tecnologia cross-platform in grado di aprire una comunicazione tra i due end-point. Inoltre questa tecnologia deve essere in grado di far eseguire al Raspberry i compiti che l'utente aziona da remoto usando il sito internet stesso. Un altro requisito fondamentale è che questa tecnologia deve essere semplice da configurare ed efficiente da un punto di vista computativo, dato che bisogna installarla sul raspberry, decisamente meno potente di un computer o uno smartphone. In letteratura esistono due tecnologie per realizzare tutto ciò: NodeJs e Python.

2.1 NodeJs

NodeJs è un'applicazione per scrivere codice client-server totalmente in Javascript sfruttanto tutte le potenzialità della programmazione asincrona *event-based*. Essendo *single-threaded*, utilizza un pattern observer per gestire tramite un solo processo decine di migliaia di connessioni, rendendolo ottimo per realizzare applicazioni DIRT (*D*ata *I*ntensive *R*eal *T*ime). NodeJs è una tecnologia con alla base

V8 Engine, un framework open-source creato da Google in grado di compilare codice Javascript in modo molto efficiente. Questo perchè utilizza delle tecniche in grado di compilare il codice in real-time, direttamente in linguaggio macchina, senza nessun processo intermediario di interpretazione .
Una libreria molto importante che usa NodeJs è *libuv*, scritta in linguaggio C che da supporto ad operazioni di tipo *asynchronous I/O* basate su event loop.

2.2 Python

Python è un linguaggio che sta diventanto sempre più importante nel mondo dell'informatica, ed è nativamente installato nel Raspberry. Nonostante non sia efficace come NodeJs per fare codice client-server, è comunque fondamentale per utilizzare al meglio la camera del raspberry attraverso la libreria *picamera* e per usare *opencv*, al momento la libreria migliore nel mercato dell'open source per fare computer vision.

Studiando sia NodeJs che Python, la soluzione migliore è stata quella di integrarli entrambi, proprio perchè il secondo è fondamentale per utilizzare OpenCv e fare Computer Vision, mentre il primo è perfetto per fare applicazioni client-server anche in real time.

2.3 OpenCV

OpenCV è l'unica libreria del mercato dell'open source che permette di fare operazioni di computer vision sfruttando funzioni e algoritmi, implementati e ottimizzati. Questa libreria è stata utilizzata per risolvere il problema della motion detection: il raspberry deve capire che c'è stato il passaggio di persone, registrando un video che terrà momentaneamente in memoria In letteratura ci sono due modalità sfruttate per risolvere la questione:

- BackgroundSubtractorMOG: questo algoritmo reisce a creare una segmentazione dell'immagine sfruttando una Gaussian Mixture, separando il background dal foreground. Questa funzione implementa il lavoro di ricerca presentato nel paper di P. KadewTraKuPong e R. Bowden, in cui si modella

ogni pixel del background con una mixture di Gaussiane che rappresentano il tempo in cui quel pixel rimane nella scena.

- Sottrazione dell'immagine. Questo metodo consiste in una pura sottrazione tra il frame contenente il modello del background e il frame corrente.

Tra i due metodi, è stato preferito il secondo perchè provandoli entrambi sono stati raggiunti risultati più consistenti, nonostante contenga difetti più difficili da risolvere, come il problema del cambiamento improvviso dell'illuminazione della stanza.

3 Analisi dei requisiti

In questa sezione esporre brevemente i requisiti a cui il sistema proposto deve rispondere, concentrando l'attenzione sugli aspetti più rilevanti e facendo eventualmente uso di opportuni diagrammi di alto livello.

Vincoli circa la lunghezza della sezione (escluse didascalie, tabelle, testo nelle immagini, schemi):

	Numero minimo di battute	Numero massimo di battute
1 componente	4000	6000
2 componenti	6000	8000
3 componenti	8000	10000

3.1 Gestione Utente

L'utente deve avere la possibilità di registrarsi al sito, con obbligatoria password, e-mail e nome utente. E-mail e nome utente devono essere unici, quindi è necessario controllare se i dati inseriti sono corretti. Inoltre, le informazioni personali devono essere usati per connettere il raspberry al sito.

3.2 Gestione Device

Una volta entrato nel sito, l'utente deve registrare per il proprio account i propri device, specificando nome, luogo in cui verrà posizionato e tipo di azione. Per informare l'utente della correttezza dell'operazione, un messaggio di feedback deve essere fornito. Deve essere fornita un'altra pagina web in cui è possibile gestire i device: l'utente dovrà vedere una lista di tutti i raspberry registrati assieme ai relativi dettagli e con i pulsanti per utilizzarli da remoto.

3.3 Operazioni con i Device

Le operazioni possibili con i device sono:

1. Accensione e spegnimento del device da remoto: deve essere possibile accendere e spegnere il device da remoto, rendendo quindi impossibile attivare le

funzioni base del device. L'utente deve capire al volo se il device è acceso o spento, pertanto un feedback deve essere garantito sull'attuale stato del componente.

2. Accensione e spegnimento delle luci: l'utente deve essere in grado di capire che l'azione è proprio quella di gestione delle luci con eventuale immagine, senza confondersi con l'azione della gestione del video. Un bottone di tipo toggle deve essere provvisto in modo che sia chiaro lo stato attuale delle luci, se spente o accese.

3. Attivazione del video: in questo modo l'utente deve essere in grado di avviare la motion detection per rivelare il passaggio delle persone. Se c'è un movimento consistente deve attivare la luce della stanza e registrare un video.

3.4 Raspberry

Il raspberry deve essere configurato manualmente dall'utente. Prima di poter utilizzarlo, è necessario che venga autentificato, collegandolo al sito nell'area personale dell'utente. Deve quindi avere un modulo di registrazione in cui l'utente deve fornire le proprie credenziali e il nome del device. Se tutto è corretto, deve comparire un messaggio di avvenuta registrazione. Lo stato del device deve essere comunicato all'utente: se si è disconesso, deve comparire un feedback preciso.

1. Raspberry con le luci: l'accensione delle luci deve essere fatta in real-time: quando l'utente spinge il bottone dal sito, allo stesso tempo viene eseguita l'azione di illuminazione. Ogni raspberry se specificato dall'utente controlla una sola luce.

2. Raspberry con la camera: il device munito di camera deve continuamente registrare un video, e quando capisce che c'è del movimento deve salvarlo in memoria, per poi inviarlo al server.

4 Progettazione

Devono essere esposte le scelte progettuali operate nelle varie fasi di sviluppo dell'elaborato.

In questa sezione devono essere documentati gli schemi di progetto relativamente all'architettura complessiva del sistema e alle sue componenti di rilievo che possano meritare un'analisi di dettaglio. Per le componenti software si può ricorrere ad esempio a diagrammi delle classi, di sequenza, stato, attività. Per le componenti hardware è possibile includere opportuni schemi in grado di descrivere l'architettura fisica adottata.

Vincoli circa la lunghezza della sezione (escluse didascalie, tabelle, testo nelle immagini, schemi):

	Numero minimo di battute	Numero massimo di battute
1 componente	9000	18000
2 componenti	12000	21000
3 componenti	15000	24000

La progettazione, data la scelta delle tecnologie, segue un'impostazione client-server attraverso la quale vengono gestiti in real-time o con regolare http request tutti gli end-point del sistema.

4.1 Server

Il server gestisce in background le scelte dell'utente, direziona le comunicazioni tra i vari client e si occupa del salvataggio dei dati sul database. Sono individuabili vari moduli distinti, il real-time, la gestione dei video tramite http post e get request e il database handling.

4.1.1 Real-Time

Il real-time è stato adottato per tutte quelle parti del sistema che richiedono un feedback istantaneo, in modo da ottenere una buona user experience nell'utilizzo delle funzionalità di maggior impatto visivo, come ad esempio l'attivazione delle

luci. Questo modulo sfrutta la tecnologia ad eventi di NodeJs per controllare le comunicazioni tra i vari end-point: utente e rete di raspberry. Il server "reagisce" alle azioni dell'utente, svolgendo un ruolo attivo solamente all'arrivo degli eventi. A livello progettuale possono essere identificati vari tipi di situazioni che il server deve rispondere:

1. Evento di accesso al sito internet. L'utente deve accedere al sito internet (mediante log-in) prima di poter utilizzare i suoi dispositivi, ma allo stesso tempo i dispositivi stessi devono essere autenticati. Questo perchè utilizzando i dati dell'autenticazione, il server crea un collegamento unico tra la persona e i suoi raspberry.

2. Evento di cambiamento di stato del device. Lo stato del device viene costantemente monitorato dal server, e in caso di disconnessione deve aggiornare il client browser per informare l'utente del problema, fornendo un feedback preciso.

3. Evento di azionamento delle funzionalità. Il server riceve l'evento di azionamento delle luci o del video e deve comunicarlo al device corretto di quell'utente, evitando quindi che altri raspberry collegati al sito non attivino le luci o il video per sbaglio.

4.1.2 Video handling

Il raspberry se abilitato è in grado attraverso la computer vision di registrare un video in cui è avvenuto il passaggio di una persona. Questo video dovrà essere mandato al server in modo che quest'ultimo possa servirlo al client browser, e viene tutto gestito attraverso il routing delle chiamate http. Il server ha essenzialmente due compiti da questo punto di vista:

1. Gestione del filesystem. I video possono arrivare in qualunque momento anche contemporaneamente da vari client ed è necessario tener traccia di ognuno di loro in modo da fornirli quando richiesto. Il server crea quindi un filesystem in cui ogni utente ha vari folder dove verrano salvati i video in modo da migliorare il tempo di ricerca quando dovrà fornirli al browser client

2. Invio dei video. Grazie al filesystem, il server riesce a trovare subito i video inviandoli via http response al browser client.

4.1.3 Database handling

Il server gestisce il database per salvare tutti i dati relativi ad ogni utente, e a sua volta viene interrogato per visualizzare i dettagli dei dispositivi a richesta dell'utente. Il database viene constantemente aggiornato per a seconda delle interazioni con l'utente e delle sue scelte, e grazie al modulo real time il risultato è aggiornato immediatamente.

4.2 Client

Gli end point del sistema sono di due tipi: il client browser e il client raspberry. Attraverso il server, dispositivi e browser comunicano per eseguire le azioni dell'utente, svolgendo anche funzionalità indipendenti mediante canali unilaterali.

4.2.1 Client Browser

Il client browser è l'interfaccia con cui vengono gestiti tutti i dispositivi di un utente, ed è la grafica stessa a suggerire all'utente cosa deve fare per attivare le varie funzionalità. Attraverso il client browser, l'utente crea un collegamento con i suoi raspberry mediato dal server, utilizzando una logica ad eventi:

1. Evento di autenticazione. L'utente inserisce i dati per ottenere un accesso al sito gestito dal server che può rifiutare il log-in se email e password non sono corretti.

2. Evento di registrazione di un device. Attraverso il client è possibile salvare i propri dispositivi online fornendo nome e tipo di funzione. Questo passo è fondamentale perchè grazie a questi dati il server riuscirà a trovare i dispositivi e successivamente unire i due end-point.

3. Evento di attivazione delle funzionalità. Proprio dal client browser parte l'evento che attiverà la funzione del dispositivo, che verrà gestito dal server.

Il client è stato progettato anche da un punto di vista grafico. L'interfaccia è importante per capire se le proprie operazioni sono andate a buon fine e grazie alla

logica ad eventi è possibile avere un feedback in real time. L'utente riesce a capire cosa non funziona e cosa sta succedendo in modo istantaneo.

1. Attivazione delle funzionalità
2. Stato del dispositivo
3. Richiesta e arrivo di un video
4. Registrazione di un dispositivo

4.2.2 Client Raspberry

Il client raspberry è progettato in due unità distinte: l'autenticazione e la gestione dell'hardware. Il primo modulo è fondamentale per creare il raspberry come un'unità completamente indipendente dall'utente, mentre il secondo è necessario per gestire la parte di computer vision con il video oppure l'accensione delle luci.

Autenticazione L'autenticazione ha due scopi: garantire flessibilità nell'usare il raspberry, o per accendere le luci o per usare il video, e per creare un collegamento unico con l'utente usando il server come tramite. In questa fase l'utente inserisce le proprie credenziali fornendo anche il nome del device, lo stesso che ha usato nel registrare il dispositivo usando il client browser, con il risultato di avere assolutà libertà nel scegliere quale raspberry usare per accendere/spegnere le luci o per fare video sorveglianza. Una volta che le informazioni provviste sono state verificate, ci penserà il server a come collegare il dispositivo al sito per svolgere le funzioni che l'utente attiverà utilizzando l'interfaccia grafica del browser.

Gestione della luce La gestione della luce funziona con una logica ad eventi che parte dal client browser: l'utente attiva dall'interfaccia l'accensione delle luci, e attraverso il server il raspberry reagisce all'azione attivando il circuito elettrico a cui è collegato. Durante questa fase, il raspberry è responsabile nel comunicare al server lo stato delle luci, aggiornando il database se le luci sono spente o accese, permettendo all'utente di avere completo controllo.

Gestione del video Questa unità lavora in modo prevalentemente indipendente rispetto al server e all'azione dell'utente: una volta che arriva l'evento di attivazione del video, la camera continua ad osservare la stanza per un movimento, analizzando ogni frame. Se viene rivelato il passaggio di una persona, il video viene inviato al server e il raspberry continua nella sua azione di motion detection.

4.3 Hardware

L'hardware utilizzato nel progetto riguarda essenzialmente il raspberry, a cui è stato collegato il circuito elettrico per attivare il led e la camera.

Circuito elettrico per accendere il led

Camera per il raspberry

4.4 Analisi in dettaglio dell'architettura event-driven

La parte di real-time è stata affrontata con una *event-driven architecture*, un tipo di architettura basata su eventi. Un evento consiste in un cambio significativo di stato del sistema o di un suo componente al quale seguiranno delle reazioni da parte di altre componenti. Questo permette di creare un sistema distribuito in grado di scalare orizzontalmente e in cui si possono attivare servizi loosely coupled rispetto le altre unità. Essendo un progetto client-server, l'architettura event-driven funziona benissimo per gestire i vari tipi di client che interessano l'intero applicativo.

5 Implementazione

Esporre i principali problemi affrontati durante l'effettiva realizzazione delle componenti hardware/software e illustrare le soluzioni implementative adottate. Se l'elaborato ha previsto l'utilizzo di tecnologie già disponibili sul mercato, discuterne brevemente le caratteristiche e motivarne l'adozione rispetto ad altre soluzioni assimilabili.

NOTA: in questa sezione devono essere riportate esclusivamente le porzioni di codice ritenute particolarmente significative. Il codice sorgente nella sua interezza, opportunamente commentato, deve essere consegnato separatamente dalla relazione in un archivio compresso.

Vincoli circa la lunghezza della sezione (escluse didascalie, tabelle, testo nelle immagini, schemi):

	Numero minimo di battute	Numero massimo di battute
1 componente	5000	11000
2 componenti	8000	16000
3 componenti	10000	21000

5.1 Problema del cross-platform e del real-time

Cross-platoform Smarthome è un applicativo web che coinvolge due tipi di client: il raspberry e il client browser. Questo significa che il primo problema da risolvere è stato la scelta di una tecnologia client-server cross-platform, che possa essere usata in modo efficiente per gestire entrambi i tipi di end-point.
NodeJs in questo campo è la scelta migliore grazie alla sua architettura event-driven, un modo di scrivere codice tutto basato su event-emitter (oggetti che emettono eventi) e event-handler (una funzione o metodo che contiene delle istruzioni eseguite in risposta ad un evento). Il primo passo del progetto è stato creare un web server e NodeJs permette di farlo sfruttando `express`, una libreria di NodeJs con varie API usate anche per semplificare il routing delle chiamate `http`.

```
var express = require('express');
var app = express();
var path = require('path');
var port = process.env.PORT || 3000;
var server = require('http').createServer(app);
//con questa linea di codice, verranno servite le pagine html
   presenti nella cartella "public" del progetto, partendo da
   index.html
app.use(express.static(path.join(__dirname, 'public')));
//creazione del server, che ascolta sulla porta 3000 (default)
   oppure su un'altra gestita dall'ambiente di hosting
server.listen(port, function () {
  console.log('listening on *:' + port);
})
```

Real-time Un obiettivo primario che è stato implementato nel progetto è il real-time, utilizzato per aggiornare il database, aggiornare l'interfaccia grafica del client browser e per attivare le funzioni dei raspberry. Utilizzando NodeJs come tecnologia, la libreria più conosciuta e usata per fare programmazione web real-time è Socket.IO. Socket.IO permette la creazione di comunicazioni bidirezionali e in real time basate su eventi per ogni tipo di piattaforma, device o browser. Socket.IO fornisce API per la creazione di socket server e socket client.

Socket server Il socket-server consiste nella creazione di una nuova istanza della socket a cui ci si potranno collegare i vari client del sistema. La socket-server gira su un'istanza del web server che viene creato nel punto precedente.

```
//nuova istanza di una socket-server
var io = require('socket.io')(server, { wsEngine: 'ws' });
//creazione di un event-listener per il socket server con il suo
   handler
io.on('connection', function (socket) {
  console.log('user connected ' + socket.id)
  }
```

In questo esempio socket.io sfrutta la tecnologia ad eventi di NodeJs, istanziando un event-listener con `.on(...)` sull'oggetto `io`, a cui segue il nome dell'evento e la

funzione handler che eseguirà varie istruzioni software. L'oggetto socket.io grazie al listener riceverà l'evento di avvenuta connessione ogni volta che un socket-client si connetterà al server, e utilizzando `function(socket)` si può avere un riferimento proprio alla socket appena collegata, su cui verranno eseguite varie istruzioni.

Socket-client La socket-client è l'end-point grazie al quale avviene una comunicazione in real-time. Questo oggetto non viene usato nel server ma inserito nel codice javascript del client, *qualunque esso sia*. Infatti si può istanziare una socket-client allo stesso modo sia per una pagina web che per un raspberry, rendendo la libreria perfetta per Smarthome.

```
1  //inclusione del file script per il client nel codice html
2  <script src="/socket.io/socket.io.js"></script>
3  <script>
4  //istanza della socket-client
5    var socket = io();
6  //creazione di un listener sulla socket-client. Una volta
       connessa attiva l'evento 'connect' e l'handlar scrive su
       console
7  socket.on('connect', function () {
8
9      console.log("I'm connecting " + socket.id)
10 })
11 </script>
```

La creazione e uso di una socket-client avviene nella stessa maniera, ma con delle differenze:

1. Parametri di connessione al server. La socket-client ha bisogno di sapere dove si trova il server nodejs per potersi connettere alla socket-server, ma si può omettere nel caso del client browser, perchè l'indirizzo viene dedotto automaticamente. Nel caso del raspberry, non si collega via browser e deve quindi specificare l'indirizzo ip o l'url del server, direttamente nei parametri della socket-client

```
1  //url dell'indirizzo ip durante il testing nel localhost
2  var stringUrl = "http://192.168.1.241:3000"
3  //url del sito web dato dal server provider (heroku)
```

```
4 var stringUrl = "https://smartsecurityhome.herokuapp.com"
5 //connessione alla socket server
6 var socket = io(stringUrl, { transports: ['websocket'] })
```

2. Riferimento alla socket-client. Un'altra differenza è che la funzione handler in `socket.on('connect', function (){})` non riceve nessun riferimento alla socket. Infatti non esiste nessun modo per un client collegarsi ad un altro client, quindi una volta istanziata la socket-client può essere usata per emettere eventi o riceverli.

Comunicazione bilaterale tra una socket-client e una socket-server La comunicazione tra una socket-server e una socket-client avviene mediante l'utilizzo dei listener con i vari handler e con l'emissione di eventi. In NodeJs, un event-emitter è un oggetto che emette un evento con un vero e proprio nome, e l'eventuale listener eseguirà un insieme di istruzioni software specificate dal programmatore. Nel codice seguente abbiamo un esempio di comunicazione tra una socket-client e una socket-server.

```
1  //codice socket-server, che gira in un istanza del web server
       creato con express
2  var io = require('socket.io')(server, { wsEngine: 'ws' });
3
4  io.on('connection', function (socket) {
5    console.log("Hi, im the socket-client with id: "+socket.id)
6    socket.emit('handshake from server',{message: "Hi
       socket-client!"})
7
8    socket.on('handshake from client', function(data){
9        var messageFromClient = data.message
10         console.log("socket-client said: "+ messageFromClient)
11     })
12
13   }
14
15
16 //socket-client, questo codice si trova nel client browser
17  var socket = io()
18
```

```
19 socket.on('handshake from server', function(data){
20  var messageFromServer =  data.message
21    console.log("server said: " + messageFromServer)
22    socket.emit('handshake from client', {message: "Hi
          socket-server!"})
23 })
```

In socket.io la comunicazione funziona in semplici passi:

1. Connessione tra socket-client e socket-server. Non può esistere una comunicazione se i due end-point non sono connessi. Osservando gli esempi precedenti, se una socket-client si congiunge alla socket-server, viene attivato l'evento `'connect'` a lato server, presente di default e dato dalla libreria.

2. Emissione di eventi. Avvenuta la connessione, a lato server viene attivato l'evento `connect`, e tocca all'handler gestire la reazione. In questo esempio viene emesso un evento creato dal programmatore `"handshake from server"` (una normale stringa), usando il metodo `emit`. Oltre all'evento è possibile inviare un messaggio in formato `JSON` che può essere manipolato dal listener della socket-client.

3. Ricezione dell'evento da parte del client. In questo momento l'evento è arrivato alla socket-client, perchè è stato definito un listener apposito per `"handshake from server"` con lo stesso nominativo. L'handler riceve il messaggio da manipolare in `data` come argomento dell'handler, stampandolo a console. A questo punto c'è un'altra emissione di un evento `"handshake from client"`, che verrà gestito alla stessa maniera dal server.

Esempio di comunicazione con l'accensione delle luci Una volta compreso il meccanismo in generale, è giusto implementarlo nel constesto di smarthome, sfruttando l'esempio più semplice dell'accensione delle luci:

1. Emissione dell'evento dal client browser. Quando l'utente preme il bottone per l'accensione (o spegnimento) delle luci, viene emesso un evento "toggle light" inviando un messaggio al server con alcune specifiche.

```
1 //socket-client del browser gia connessa
2 var socket = io()
```

```
3  //funzione handler jquery per il toggle del bottone
4   if (actiontype == 'light') {
5    $('#' + uniqueID + ' input').change(function () {
6
7      if (this.checked) {
8  //il client browser emette un evento 'toggle light'
9        socket.emit('toggle light', { devicename: deviceName,
             light: true, username: user.username })
10         }
11     else {
12       socket.emit('toggle light', { devicename: deviceName,
             light: false, username: user.username })
13       }
14   })
```

2. Ricezione dell'evento da parte del server. Con un apposito handler avente lo stesso nome (`"toggle light"`), il server emette `"turn on/off light"`, che verrà gestito da quelle socket-client con l'handler corretto.

```
1  //handler dell'evento "toggle light" del socket-server
2  //in questo esempio viene tralasciato l'aggiornamento dello
       stato delle luci sul database
3  socket.on('toggle light', function (data) {
4      console.log('light toggled ' + socket.id)
5      //emesso dal server
6      socket.emit('turn on/off light', { light: actionstatus,
           devicename: devicename })
7    })
```

3. Handling dell'evento da parte del raspberry. Questo handler, utilizzando i dati contenuti del messaggio, utilizza una libreria di NodeJs (`onoff`) per attivare il voltaggio del circuito elettrico del LED.

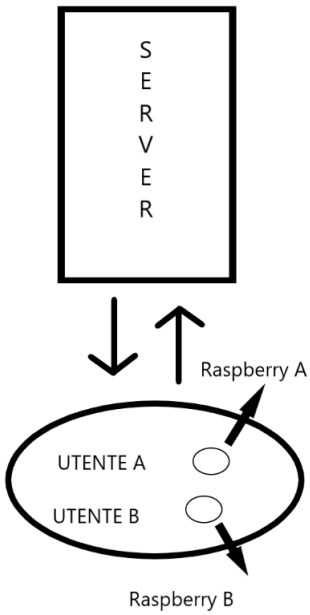
```
1   //socket-client del raspberry con il relativo handler
2  socket.on('turn on/off light', function (data) {
3    if (LED.readSync() === 0 && data.light && deviceName ===
       data.devicename && on) {   //check the pin state, if
       the state is 0 (or off)
4       console.log('data arrived: ' + data.light)
```

```
5           LED.writeSync(1); //set pin state to 1 (turn LED on)
6           console.log('turning on light')
7       }
8       else if (deviceName === data.devicename) {
9           LED.writeSync(0)//set pin state to 0 (turn LED off)
10          console.log('turning off light')
11      }
12  })
```

5.2 Problema della creazione di un sistema distribuito e multi-utente

Fin dall'inizio *Smarthome* è stato progettato da un punto di vista del sistema distribuito, con più utenti che possono usarlo contemporaneamente.

Primo tentativo per gestire la multi-utenza Lo stesso codice dei vari handler è presente in ogni client e in un primo tentativo non c'è nessuna distinzione tra i vari utenti con i loro raspberry. Consideriamo l'esempio dell'accensione delle luci: dato che ogni socket-client è connessa al server, qualunque raspberry con l'handler per `"turn on/off light"` gestirà l'evento.



Dato che il server non fa distinzioni, se l'utente A emette l'evento `"toggle light"`, il server manderà l'evento `"turn on/off light"` anche ai raspberry dell'utente B, nonostante il loro vero proprietario non abbiato attivato nessuna funzionalità.

Creazione di connessioni uniche tra utente e i suoi raspberry con il server La soluzione implementata consiste nell'usare un sistema caratteristico di socket.io: le *room*. Una *room* è una vera e propria stanza in cui ogni evento emesso rimane confinato tra le socket che si sono unite. Il risultato che si vuole ottenere è che se il server invierà un evento nella *room* dell'utente A, solo i listener dei raspberry di A attiveranno il relativo handler. La creazione di una *room* può essere fatto in modo dinamico al momento della connessione del socket-client al socket-server, utilizzando i dati che vengono forniti al momento dell'autenticazione.

```
1 //la socket si connette al server
2 var socket = io({ transports: ['websocket'] });
3 //nel local storage dell'applicativo web viene memorizzato
     l'username dell'utente
4 var user = JSON.parse(localStorage.getItem('currentuser'))
     //properties: username
5 socket.on('connect', function () {
6 //utilizzando i dati del local storage, viene inviato un evento
     con un messaggio assieme allo username dell'utente
7     console.log("I'm connecting " + socket.id + " to room " +
         user.username)
8     socket.emit('room', { username: user.username})
9 })
```

Una volta aperta la pagina web con la lista dei dispositivi, il client emette un evento `"room"` con il nome dell'utente che verrà usato a lato server per creare una stanza. Le socket-client devono sempre avere nel messaggio dell'evento il nome, altrimenti il server non sa a quale stanza rispondere (notare infatti che nell'esempio dell'accensione delle luci nel messaggio c'è sempre un riferimento allo username).

```
1 //listener del server sull'evento room
2 socket.on('room', (data) => {
3     var roomName = data.username
4     //creazione della room con il nome utente
5     socket.join(roomName)//socket joins room with id of username
```

```
6       //esempio di invio evento solo nella room appena creata
7       io.in(roomName).clients(function (error, clients) {
8         if (error) throw error
9         console.log(clients)
10      })
11      socket.emit('room joined', { roomjoined: roomName, id:
            socket.id })
12
13    })
```

L'autenticazione deve avvenire anche per il raspberry: prima di essere utilizzato è necessario che si colleghi alla stanza del suo proprietario, alla stessa maniera con cui l'utente effettua il log-in. Sfruttando i dati forniti, la socket-client del raspberry invierà evento room usando i dati ricevuti. Il modulo di autenticazione per il raspberry è stato realizzato con la libreria `inquirer`, che permette di stampare a console delle domande memorizzando le risposte fornite da usare in seguito. Innanzitutto viene creato un collegamento tra computer e raspberry via `ssh` utilizzando `putty`, un software per `Windows`. Appena il codice viene compilato, su console l'utente può dare i propri dati: nome utente e password (per controllare se l'utente esiste e se si è già registrato) con anche il nome del device.

```
pi@raspberrypi:~/Documents/smarthome $ node rpiclient.js
connected
LFInNyu33w0lT_OnAAAA
? E-mail:  j@m.com
? Password:  [hidden]
? Device name:  video kitchen
logging in Jack
username registered!  Jack
```

Una volta controllato che l'utente esista, la socket-client del raspberry si può unire alla stessa stanza della socket-client browser.

5.3 Problema della motion-detection

Un servizio che *Smarthome* deve fornire è *motion-detection*, un processo che coinvolge il modulo della camera per comprendere se è passato qualuno nella stanza. Nel caso in cui avviene il passaggio, viene salvato in memoria un video per poi spedirlo al server. Nel campo della computer vision la libreria open-source migliore è OpenCV, le cui funzioni permettono di analizzare in modo approfondito molti aspetti della visione artificiale. Un altro vantaggio nell'usare python per OpenCV consiste nell'utilizzo efficiente anche di `picamera`, una libreria molto potente scritta in python per fare operazioni con la camera.

Interfacciamento di uno script di opencv con nodejs Opencv è una libreria che può essere usato o in `C++` oppure in `Python`, creando il primo problema dell'interfacciamento dello script con nodejs. Grazie alla libreria `PythonShell`, è possibile compilare uno script `python` e anche mandare messaggi utilizzando codice `Javascript`

```
socket.on('turn on/off video', function (data) {
  //compilazione dello script python
    var shell = new PythonShell('camerascript.py', options)
    if (data.video && deviceName === data.devicename && on) {
        console.log('data arrived: ' + data.video)
        console.log('turning on video')
        //invio di un messaggio
        shell.send('start recording')
})
```

Possibili modi per fare motion detection Tra le varie soluzioni possibili, due possibili sono state scelte: riconoscimento dei volti e *background subtraction*. Il primo tentativo è stato sfruttare il riconoscimento del viso, utilizzando OpenCV per addestrare un classificatore e utilizzarlo per riconoscere i volti in nuovi frame, forniti dalla camera. OpenCV fornisce metodi e classi per la classificazione, partendo da degli algoritmi che utilizzano le immagini del training set per l'addestramento, fornendo come risultato un file xml usato dal classificatore finale. Il classificatore per analizzare i volti nei frame deve essere supportato dal `face detector`, in grado di trovare il volto nel frame. Una volta trovato il volto, signi-

fica che c'è stata motion detection e nel video registrato si può utilizzare alcune utilities di OpenCV per marcare l'area di interesse e passarla al classificatore informando l'utente chi è passato. Questo tentativo è stato tradotto in codice nel file `facescript.py`, ma con risultati poco concludenti. La maggior parte delle volte il `face detector`, presente di default in OpenCV, non riesce a trovare il volto risultando impraticabile il tentativo di classificazione.

Motion detection by background subtraction Questo metodo è concettualmente più semplice e più immediato: utilizzando un frame come modello dell'ambiente, si può fare la sottrazione tra il frame corrente e il primo, e analizzare il risultato sfruttando il *thresholding*, una tecnica che permette di creare immagini binarie se l'intensità del pixel è sopra o sotto una certa soglia.

```
1  //con picamera viene preso il frame corrente durante il video
2  current_frame = cv2.cvtColor(rawCapture.array, cv2.COLOR_BGR2GRAY)
3  //se nessun frame e' stato scelto come modello, allora il primo
      preso diventa quello di riferimento
4  if firstFrame is
5  None or updateBackgroundModel(timeFirstFrame):
6          print('updating background model')
7          firstFrame = current_frame
8  //differenza tra i due frame
9  frameDelta = cv2.absdiff(firstFrame, current_frame)
10 //thresolding della differenza
11 thresh = cv2.threshold(frameDelta, 20, 255, cv2.THRESH_BINARY)[1]
12 //tecnica di dilatazione
13 thresh = cv2.dilate(thresh, None, iterations=2)
```

In questa prima parte dell'algoritmo, vengono effettuate delle operazioni preliminari per poter capire se c'è stato del movimento:

1. Cattura del frame corrent. Grazie alla libreria `picamera` è possibile scattare delle foto durante la ripresa del video con il vantaggio di fare analisi in tempo reale dell'ambiente.

2. Frame come modello. Il frame corrente diventa modello in due situazioni: non è stato ancora nominato nessun frame di riferimento oppure sono passati 10 secondi da quando è stato aggiornato.

3. Differenza. Utilizzando il frame di riferimento, viene effettuata una differenza utilizzando `numpy`, una libreria di python che permette di trattare le immagini come array. Una volta effettuata, il risultato è che se nel frame corrente c'è la persona, ma nel modello no, i pixel aventi la stessa o simile intesità diventano molto più scuri, mentre dove ci sarebbe la persona diventano più chiari.

4. Processing. Utilizzando la nuova immagine differenza come `numpy` array, si può effettuare il thresolding creando un immagine binaria. Notare come i pixel statici appartenenti all'ambiente diventano completamente neri mentre dove c'è la persona i pixel diventano tutti bianchi. Sfruttando anche la *dilatazione*, vengono riempiti gli spazi neri molto vicini tra i pixel bianchi per un'analisi più qualitativa.

Nella prossima parte viene fatto un conteggio dei pixel bianchi: se il numero supera una certa soglia di confidenza, significa che c'è stato del movimento e il video si interrompe, venendo spedito al server e lo stesso processo viene ripreso.

6 Testing e performance

Esporre lo stato di funzionamento effettivo del sistema progettato ad elaborato concluso. Per ciascuna delle funzionalità salienti devono essere tabellate e discusse le performance riscontrate mediante opportuni test eseguiti in fase di validazione del progetto.

I tempi di esecuzione/comunicazione devono essere accompagnati dalle caratteristiche dell'hardware sul quale è eseguito il software.

Qualora l'elaborato includa algoritmi innovativi, indicarne la complessità computazionale (avendo cura di esporre lo pseudo codice nella sezione 5).

Vincoli circa la lunghezza della sezione (escluse didascalie, tabelle, testo nelle immagini, schemi):

	Numero minimo di battute	Numero massimo di battute
1 componente	2000	3000
2 componenti	2500	4500
3 componenti	3000	6000

7 Analisi di deployment su larga scala

In questa sezione va discussa, eventualmente con l'ausilio di opportuni diagrammi (componenti, deployment), l'evoluzione del progetto presentato immaginando che venga adottato su larga scala. I dettagli qui esposti devono quindi astrarre dalle specifiche dell'elaborato qualora l'implementazione sia stata focalizzata su uno scenario isolato.

A titolo d'esempio, qualora applicabile, devono essere evidenziate le criticità che si potrebbero incontrare e devono essere proposte soluzioni tipiche in contesti di *cloud architecture* per garantire un'adeguata *resilienza*, in termini di *availability* e *scalability* del sistema.

Vincoli circa la lunghezza della sezione (escluse didascalie, tabelle, testo nelle immagini, schemi):

	Numero minimo di battute	Numero massimo di battute
1 componente	3000	6000
2 componenti	4500	9000
3 componenti	6000	12000

8 Piano di lavoro

In questa sezione devono essere chiariti i compiti svolti da ciascun candidato nel caso in cui il gruppo abbia più di un componente.

Deve essere inoltre esposto il piano di lavoro adottato. A tal fine, per ogni attività svolta durante la preparazione dell'elaborato (ad esempio: studio di una tecnologia, progettazione di un componente, implementazione di un algoritmo ecc...) deve essere chiarita la collocazione temporale e devono essere indicate le risorse impiegate per svolgerla (giorni/uomo). I candidati possono ricorrere a opportuni diagrammi come quello di Gantt.

Vincoli circa la lunghezza della sezione (escluse didascalie, tabelle, testo nelle immagini, schemi):

	Numero minimo di battute	Numero massimo di battute
1 componente	1000	2000
2 componenti	1500	3000
3 componenti	2000	4000

9 Conclusioni

Esporre brevemente le considerazioni conclusive sul progetto presentato, indicando anche i possibili sviluppi futuri.

Vincoli circa la lunghezza della sezione (escluse didascalie, tabelle, testo nelle immagini, schemi):

	Numero minimo di battute	Numero massimo di battute
1 componente	500	1000
2 componenti	1000	2000
3 componenti	1500	3000

Appendice

Laddove necessario è possibile avvalersi di appendici alla relazione per includere materiale di approfondimento.

A titolo esemplificativo possono essere incluse le schede tecniche dei componenti adottati, la normativa di riferimento che regola un particolare dominio applicativo, ecc.

Riferimenti bibliografici

Elencare i riferimenti bibliografici citati nel testo.