Guida Modulo

back-end

Capitolo Pagina

back-end

L'unione di tutti i moduli del progetto avviene nel back-end, il quale deve collegare il front-end, cioé la web-app, il database di domotics, e i micro controllor (nel nostro caso Arduino), tramite l'ACC.

Il back-end è scritto in java, e verrà servito da tomcat, come web server.

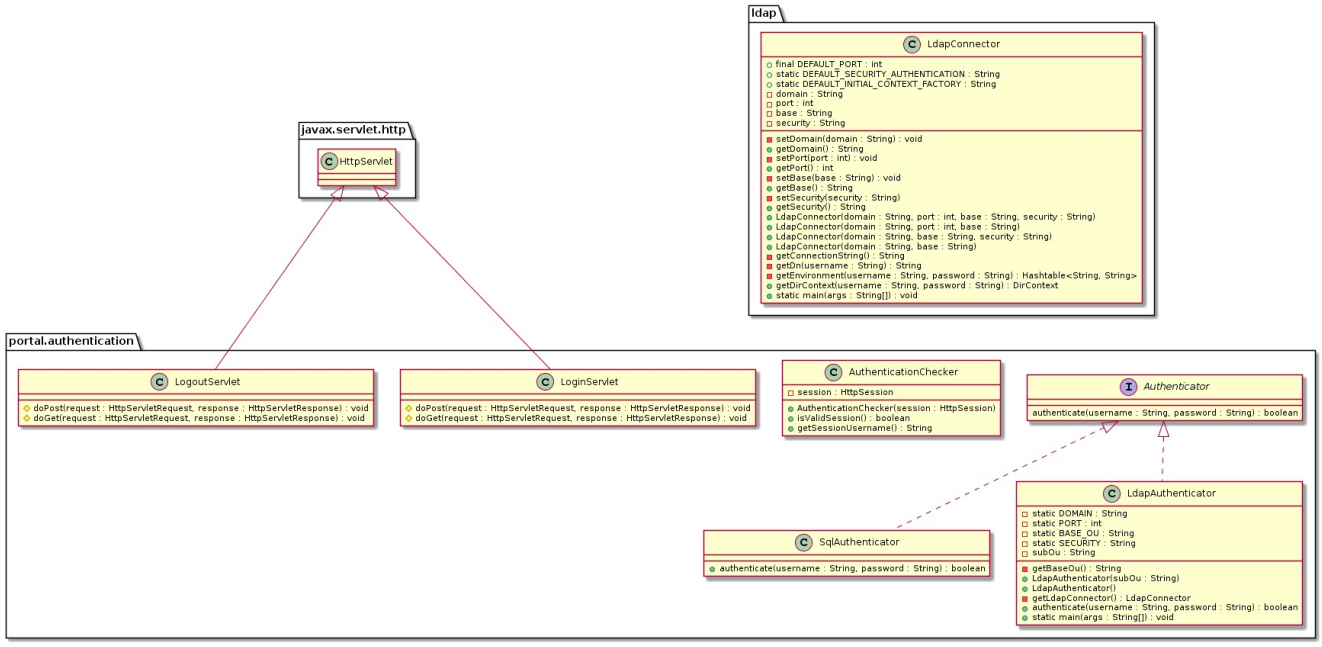
Design

Il back-end, è stato progettato in più fasi, modulo per modulo.

LDAP

Il primo modulo ad essere stato progettato è stato quello di `ldap` e dell'autenticazione, che si compone delle classi:

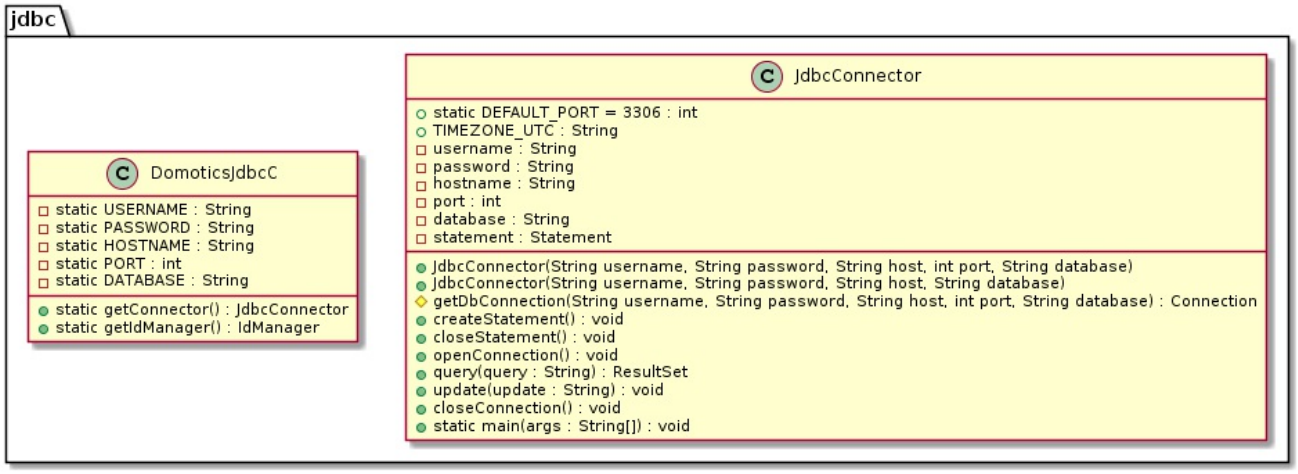
- `LdapConnector`, gestisce le connessioni con il server ldap.
- `Authenticator`, è un interfaccia creata per poter utilizzare diversi tipi di autenticazione con facilità
- `LdapAuthenticator`, classe per autenticarsi con ldap, implementa `Authenticator`.
- `AuthenticationChecker`, una classe utilizzata per convalidare le sessioni HTTP.
- `LoginServlet`, servlet per eseguire il login della sessione HTTP.
- `LogoutServlet`, servlet per invalidare la sessione HTTP.



JDBC

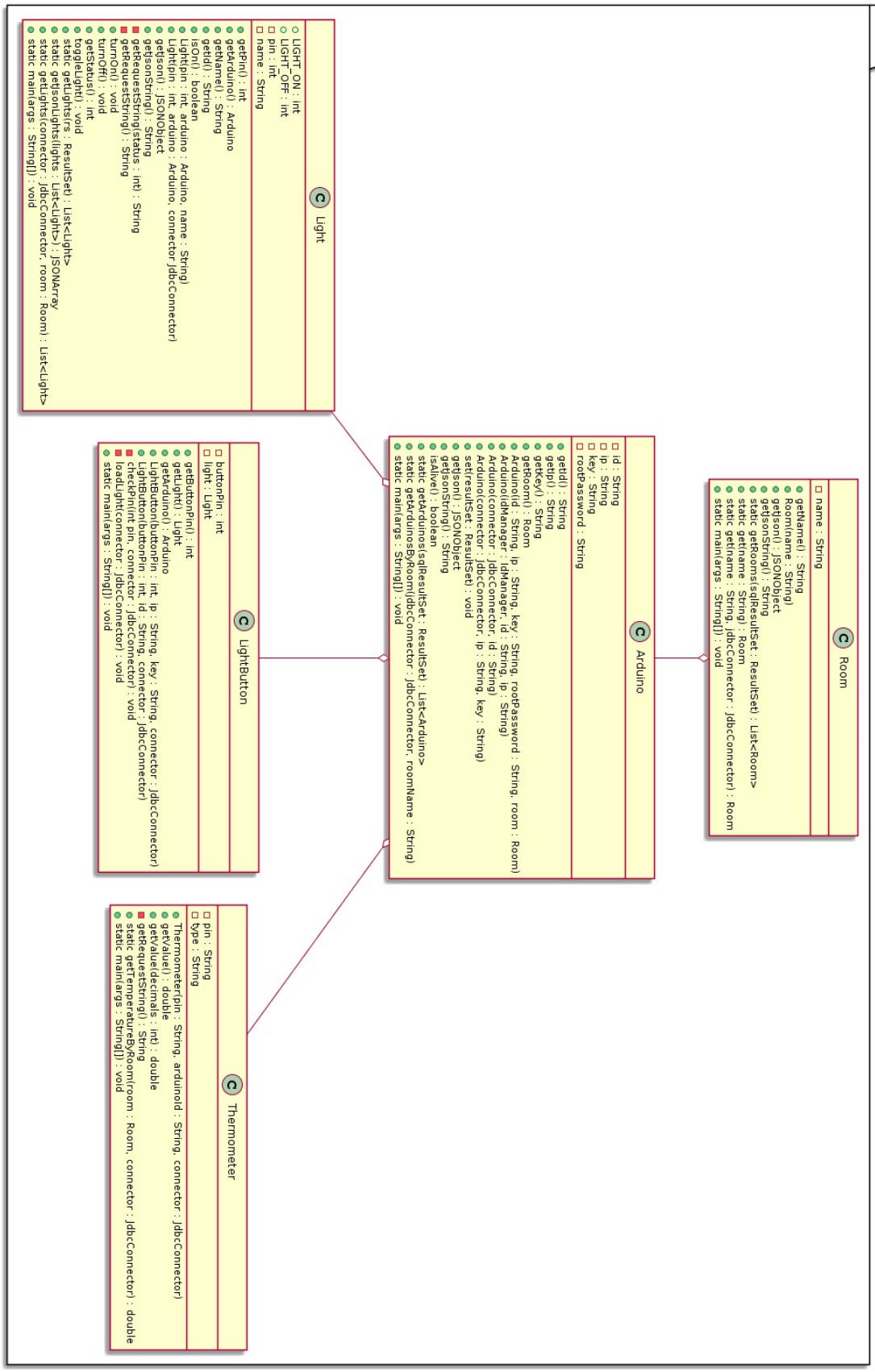
Dopo di che è stato progettato il modulo relativo al database, quindi l'utilizzo di JDBC, il driver per connnettersi ai database MySQL con Java.
Il quale è composto delle seguenti classi:

- `JdbcConnector`, gestisce le connessioni al server MySQL, per poter funzionare necessita che vi sia presente la libreria `mysql-connector-java-8.0.13.jar`.
- `DomoticsJdbcC`, è una classe che istanzia `JdbcConnector` nella maniera corretta per questo progetto.



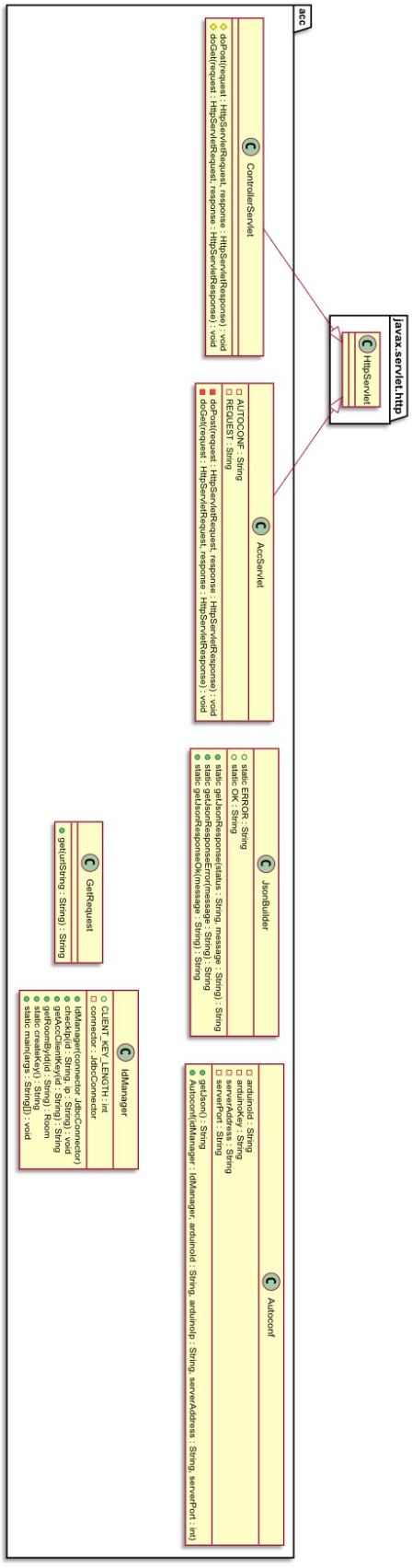
Modelli database

Dopo aver progettato il modulo della connessione al database, sono state progettate le classi che rappresentano le istanze dei database e che ci interaggiscono direttamente.
Per ogni tabella del database, è stata creata una classe. La quale servirà per aiutare l'interazione con il database e gli altri moduli. Le classi sono:


models
Light
LIGHT_ON : int
LIGHT_OFF : int
pin : int
name : String
getPin() : int
getArduino() : Arduino
getName() : String
getId() : String
isOn() : boolean
LightPin : int, arduino : Arduino, name : String)
LightPin : int, arduino : Arduino, connector:JdbcConnector)
getJson() : JSONObject
getJsonString() : String
getRequestString(status : int) : String
getRequestString() : String
turnOn() : void
turnOff() : void
getStatus() : int
toggleLight() : void
static getLights(rs : ResultSet) : List<Light>
static getJsonLights(lights : List<Light>) : JSONArray
static getLights(connector : JdbcConnector, room : Room) : List<Light>
static main(args : String[]) : void
Arduino
id : String
ip : String
key : String
rootPassword : String
getId() : String
getIp() : String
getKey() : String
getRoom() : Room
Arduino(id : String, ip : String, key : String, rootPassword : String, room : Room)
Arduino(idManager : IdManager, id : String, ip : String)
Arduino(connector : JdbcConnector, id : String)
Arduino(connector : JdbcConnector, ip : String, key : String)
setResultSet : ResultSet) : void
getJson() : JSONObject
getJsonString() : String
isAlive() : boolean
static getArduinos(sqlResultSet : ResultSet) : List<Arduino>
static getArduinosByRoom(jdbcConnector : JdbcConnector, roomName : String)
static main(args : String[]) : void
Room
name : String
getName() : String
Room(name : String)
getJson() : JSONObject
getJsonString() : String
static getRooms(sqlResultSet : ResultSet) : List<Room>
static getName : String) : Room
static getName : String, jdbcConnector : JdbcConnector) : Room
static main(args : String[]) : void
LightButton
buttonPin : int
light : Light
getButtonPin() : int
getLight() : Light
getArduino() : Arduino
LightButton(buttonPin : int, ip : String, key : String, connector : JdbcConnector)
LightButton(buttonPin : int, id : String, connector : JdbcConnector)
checkPin(int pin, connector : JdbcConnector) : void
loadLight(connector : JdbcConnector) : void
static main(args : String[]) : void
Thermometer
pin : String
type : String
Thermometer(pin : String, arduinoId : String, connector : JdbcConnector)
getValue() : double
getValue(decimals : int) : double
getRequestString() : String
static getTemperatureByRoom(room : Room, connector : JdbcConnector) : double
static main(args : String[]) : void

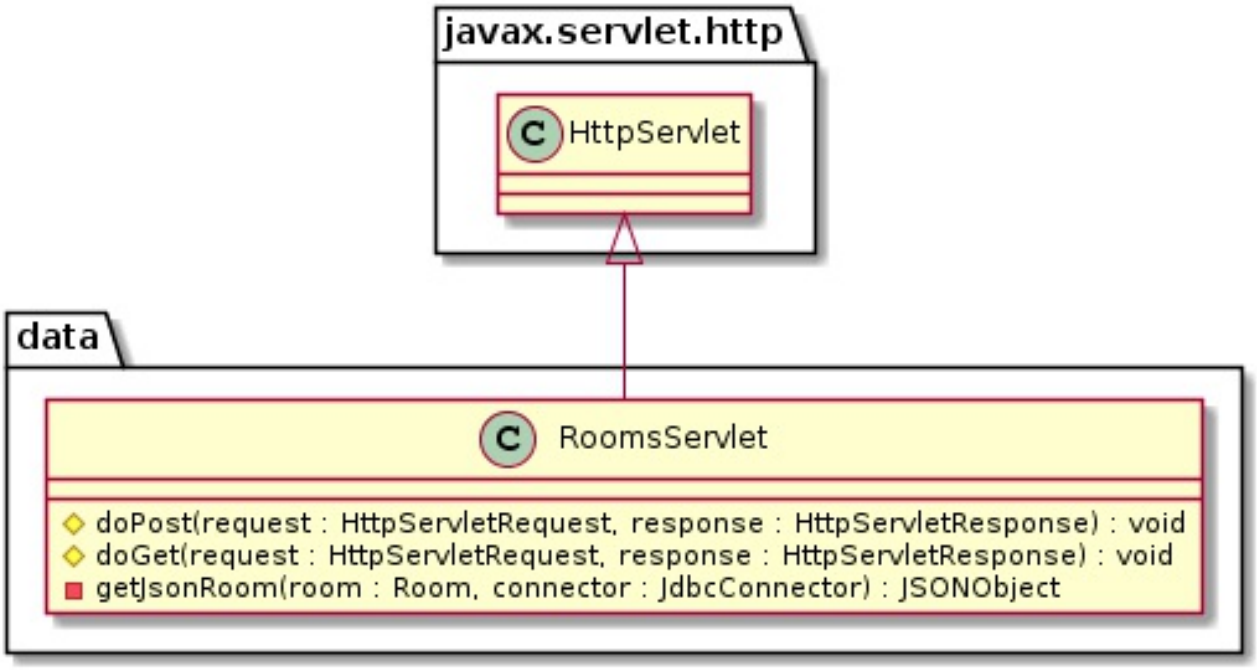
ACC-Server

Dopo aver progettato le classi modello per il database, è stato progettato il modulo dell'ACC lato server, sono state progettate le classi di cui avrebbe necessitato, per poter funzionare correttamente con gli elementi già esistenti. Queste classi sono state studiate, in maniera da mantenere i vari elementi più separati possibile, così da poter sostituire o modificare i vari elementi più facilmente possibile. Infatti le richieste vengono interpretate dalla servlet (`AccServlet`), i controlli sui micro controllori vengono eseguiti dalla classe preposta (`IdManager`), le richieste HTTP ai microcontrollori vengono eseguite tramite la classe `GetRequest` e le configurazioni per i microcontrollori vengono generate dal `Autoconf` .


acc
ControllerServlet
doPost(request : HttpServletRequest, response : HttpServletResponse) : void
doGet(request : HttpServletRequest, response : HttpServletResponse) : void
javax.servlet.http
HttpServlet
AccServlet
AUTOCONF : String
REQUEST : String
doPost(request : HttpServletRequest, response : HttpServletResponse) : void
doGet(request : HttpServletRequest, response : HttpServletResponse) : void
JsonBuilder
static ERROR : String
static OK : String
static getJsonResponse(status : String, message : String) : String
static getJsonResponseError(message : String) : String
static getJsonResponseOK(message : String) : String
Autoconf
arduinoId : String
arduinoKey : String
serverAddress : String
serverPort : String
getJson() : String
Autoconf(idManager : IdManager, arduinoId : String, arduinoIp : String, serverAddress : String, serverPort : int)
GetRequest
getUrlString : String) : String
IdManager
CLIENT_KEY_LENGTH : int
connector : JdbcConnector
IdManager(connector JdbcConnector)
createClient(name : String) : void
getAccClientKey(id : String) : String
getRoomById(id : String) : Room
static createKey() : String
static main(args : String[]) : void

Dati per front-end (JSON)

Da ultimo è stato progettato il modulo relativo ai dati, i dati da inviare al front-end. Che è composto dalla classe `RoomsServlet`, in futuro potrebbe venir anche ingrandito. Questa classe, semplicemente richiede al dababase (tramite le classi modello), tutte le `Room`, con le relative luci. E trasforma tutto in un file JSON, che viene inviato come risposta.



Implementazione

Per quanto riguarda l'Implementazione, è stato un processo, che ha preso tempo, ed è stato fatto durante la progettazione degli altri moduli, quando un modulo veniva progettato, veniva subito implementato e nel frattempo si progettava l'altro modulo. Questo perchè l'obbiettivo era mantenerli indipendenti, utilizzando questo approccio eravamo anche più liberi di scegliere l'attività da fare potendo spaziare tra progettazione ed implementazione.

LDAP

Per il modulo di LDAP la parte complicata è quella di costruire la stringa di connessione tramite la `Hashtable`.

```
/**
 * Get the hashtable environment of the connection.
 *
 * @param username Username of the connection.
 * @param password Password of the connection.
 * @return Hashtable Environment of the connection.
 */
private Hashtable<String, String> getEnvironment(String username, String password) {
    Hashtable<String, String> environment = new Hashtable<String, String>();

    environment.put(Context.INITIAL_CONTEXT_FACTORY, DEFAULT_INITIAL_CONTEXT_FACTORY);
    environment.put(Context.PROVIDER_URL, getConnectionString());
    environment.put(Context.SECURITY_AUTHENTICATION, getSecurity());
    environment.put(Context.SECURITY_PRINCIPAL, getDn(username));
    environment.put(Context.SECURITY_CREDENTIALS, password);

    return environment;
}
```

La quale viene creata in un metodo dedicato.

JDBC

Nell'implementazione del modulo di JDBC, i punti complicati, sono:

- inserire la libreria in maniera che il sistema ne possa fare uso
- creare la giusta stringa di connessione, per la quale abbiamo utilizzato un trucchetto per evitare problemi con gli orari non sincronizzati fra web-server, e server MySQL.

Il driver di JDBC (libreria) va inserito come libreria esterna del progetto (quando si lavora su IntelliJ IDEA), mentre per il server nelle librerie di tomcat.

Mentre per creare la stringa di connessione, bisogna inserire il driver (`jdbc`), il protocollo (`mysql`), l'host, la porta ed il nome del database. Infine aggiungere un `?` con la stringa dell'orario UTC.

```
final String TIMEZONE_UTC =
"useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC";
String connectionString = "jdbc:mysql://" + host + ":" + port + "/" + database + "?";
connectionString += TIMEZONE_UTC;
```

Modelli database

Per i modelli del database, sono semplicemente state riprese le tabelle del database, e scritti i relativi oggetti. Dopo di che, sono stati aggiunti alcuni metodo per eseguire più facilmente determinate operazioni.

ACC-Server

Per l'ACC-Server sono state implementate le varie classi, le sono principalmente `IdManager`, che si occupa di controllare che siano autentici gli Id, gli Ip e le key dei microcontrollori, e la classe AccServlet, che si occupa soddisfare le richieste provenienti dall'ACC-Client.

Dati per il front-end (JSON)

Per la genrazione del file JSON da ritornare al front-end come dati, la maggior parte di questo processo avviene nel metodo `getJsonRoom()` della classe `RoomsServlet`, il quale richiede lo stato di tutte le luci, la temperatura della stanza, il nome della stanza e crea un oggetto JSON, il quale verrà aggiunto ad un'array di oggetti JSON, che infine verrà ritornato come stringa al front-end.

```
private JSONObject getJsonRoom(Room room, JdbcConnector jdbc) throws SQLException,
ClassNotFoundException, IOException {
    JSONObject roomJson = new JSONObject();

    JSONArray lights = Light.getJsonLights(Light.getLights(jdbc, new Room(room.getName())));

    roomJson.put("lights", lights);
    roomJson.put("name", room.getName());
    roomJson.put("temp", Thermometer.getTemperatureByRoom(room, jdbc) + "");

    return roomJson;
}
```