**Prerequisiti**

Assicurati di avere installato i seguenti strumenti:

1. **Node.js (versione 16+)** - [Download](https://nodejs.org/)
2. **npm (versione 7+)** o **yarn**
3. **Git** - [Download](https://git-scm.com/downloads)
4. **MongoDB (opzionale per sviluppo)** - [Download](https://www.mongodb.com/try/download/community)
5. **Editor di codice** (consigliato VS Code) - [Download](https://code.visualstudio.com/)

**1. Installazione dell'Applicazione**

**1.1 Clonare il Repository**

*# Crea una directory per il progetto*

mkdir blue-site-monitor

cd blue-site-monitor

*# Inizializza un nuovo repository git (se non lo hai già fatto)*

git init

*# Copia tutti i file presenti nel repository*

*# Puoi anche clonare il repository se è su GitHub/GitLab*

**1.2 Struttura del Progetto**

Assicurati che la struttura del progetto sia come segue:

blue-site-monitor/

├── client/ # Frontend Vue.js

├── server/ # Backend Node.js

├── mock-servers/ # Server mock per sviluppo

├── .gitignore

├── package.json # Configurazione root del progetto

└── README.md

**1.3 Configurazione dei File package.json**

**Package.json Root**

Crea un file package.json nella directory root con il seguente contenuto:

{

"name": "blue-site-monitor",

"version": "1.0.0",

"description": "BlueIOT Site Monitor - A web application for real-time tracking in construction sites",

"scripts": {

"client": "cd client && npm run dev",

"server": "cd server && npm run dev",

"mock-blueiot": "cd mock-servers/blueiot-mock && npm start",

"mock-crm": "cd mock-servers/crm-mock && npm start",

"dev": "concurrently \"npm run server\" \"npm run client\" \"npm run mock-blueiot\" \"npm run mock-crm\"",

"setup": "npm install && npm run setup:client && npm run setup:server && npm run setup:mock-servers",

"setup:client": "cd client && npm install",

"setup:server": "cd server && npm install",

"setup:mock-servers": "cd mock-servers/blueiot-mock && npm install && cd ../crm-mock && npm install",

"build": "npm run build:client && npm run build:server",

"build:client": "cd client && npm run build",

"build:server": "cd server && npm run build"

},

"devDependencies": {

"concurrently": "^7.6.0"

}

}

**Package.json Client**

Crea un file package.json nella directory client:

{

"name": "blue-site-monitor-client",

"version": "1.0.0",

"type": "module",

"scripts": {

"dev": "vite",

"build": "vue-tsc --noEmit && vite build",

"preview": "vite preview"

},

"dependencies": {

"@fortawesome/fontawesome-svg-core": "^6.4.0",

"@fortawesome/free-solid-svg-icons": "^6.4.0",

"@fortawesome/vue-fontawesome": "^3.0.3",

"axios": "^1.4.0",

"events": "^3.3.0",

"pinia": "^2.1.4",

"three": "^0.153.0",

"vue": "^3.3.4",

"vue-router": "^4.2.2"

},

"devDependencies": {

"@types/events": "^3.0.0",

"@types/node": "^20.3.1",

"@types/three": "^0.152.1",

"@vitejs/plugin-vue": "^4.2.3",

"autoprefixer": "^10.4.14",

"postcss": "^8.4.24",

"tailwindcss": "^3.3.2",

"typescript": "^5.1.3",

"unplugin-auto-import": "^0.16.4",

"vite": "^4.3.9",

"vue-tsc": "^1.8.0"

}

}

**Package.json Server**

Crea un file package.json nella directory server:

{

"name": "blue-site-monitor-server",

"version": "1.0.0",

"scripts": {

"start": "node dist/app.js",

"dev": "nodemon --exec ts-node src/app.ts",

"build": "tsc"

},

"dependencies": {

"cors": "^2.8.5",

"dotenv": "^16.3.1",

"express": "^4.18.2",

"jsonwebtoken": "^9.0.0",

"mongoose": "^7.3.0",

"morgan": "^1.10.0",

"multer": "^1.4.5-lts.1",

"ws": "^8.13.0"

},

"devDependencies": {

"@types/cors": "^2.8.13",

"@types/express": "^4.17.17",

"@types/jsonwebtoken": "^9.0.2",

"@types/morgan": "^1.9.4",

"@types/multer": "^1.4.7",

"@types/node": "^20.3.1",

"@types/ws": "^8.5.5",

"nodemon": "^2.0.22",

"ts-node": "^10.9.1",

"typescript": "^5.1.3"

}

}

**1.4 Configurazione TypeScript**

**tsconfig.json Server**

Crea un file tsconfig.json nella directory server:

{

"compilerOptions": {

"target": "ES2020",

"module": "commonjs",

"outDir": "./dist",

"rootDir": "./src",

"strict": true,

"esModuleInterop": true,

"skipLibCheck": true,

"forceConsistentCasingInFileNames": true,

"resolveJsonModule": true

},

"include": ["src/\*\*/\*"],

"exclude": ["node\_modules", "\*\*/\*.test.ts"]

}

**tsconfig.json Client**

Crea un file tsconfig.json nella directory client:

{

"compilerOptions": {

"target": "ES2020",

"useDefineForClassFields": true,

"module": "ESNext",

"lib": ["ES2020", "DOM", "DOM.Iterable"],

"skipLibCheck": true,

*/\* Bundler mode \*/*

"moduleResolution": "bundler",

"allowImportingTsExtensions": true,

"resolveJsonModule": true,

"isolatedModules": true,

"noEmit": true,

"jsx": "preserve",

*/\* Linting \*/*

"strict": true,

"noUnusedLocals": true,

"noUnusedParameters": true,

"noFallthroughCasesInSwitch": true,

*/\* Paths \*/*

"baseUrl": ".",

"paths": {

"@/\*": ["src/\*"]

}

},

"include": ["src/\*\*/\*.ts", "src/\*\*/\*.d.ts", "src/\*\*/\*.tsx", "src/\*\*/\*.vue"],

"references": [{ "path": "./tsconfig.node.json" }]

}

Crea anche tsconfig.node.json nella directory client:

{

"compilerOptions": {

"composite": true,

"skipLibCheck": true,

"module": "ESNext",

"moduleResolution": "bundler",

"allowSyntheticDefaultImports": true

},

"include": ["vite.config.ts"]

}

**2. Configurazione di MongoDB**

**2.1 Installazione di MongoDB**

**Per Windows:**

1. Scarica il programma di installazione da [MongoDB Download Center](https://www.mongodb.com/try/download/community)
2. Esegui il programma di installazione e segui le istruzioni
3. Seleziona l'opzione "Complete" durante l'installazione
4. Installa MongoDB Compass (interfaccia grafica) se lo desideri

**Per macOS:**

*# Utilizzando Homebrew*

brew tap mongodb/brew

brew install mongodb-community@6.0

*# Avvia MongoDB*

brew services start mongodb-community@6.0

**Per Linux (Ubuntu):**

*# Importa la chiave pubblica*

wget -qO - https://www.mongodb.org/static/pgp/server-6.0.asc | sudo apt-key add -

*# Crea un file list per MongoDB*

echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu $(lsb\_release -cs)/mongodb-org/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list

*# Aggiorna il database dei pacchetti*

sudo apt-get update

*# Installa MongoDB*

sudo apt-get install -y mongodb-org

*# Avvia il servizio MongoDB*

sudo systemctl start mongod

*# Verifica che sia in esecuzione*

sudo systemctl status mongod

**2.2 Configurazione MongoDB per lo Sviluppo**

Per lo sviluppo, puoi utilizzare MongoDB in due modi:

1. **MongoDB locale:** installato sulla macchina come descritto sopra
2. **MongoDB Atlas:** servizio cloud gratuito (per sviluppo)

**Configurazione MongoDB Atlas:**

1. Registrati su [MongoDB Atlas](https://www.mongodb.com/cloud/atlas/register)
2. Crea un nuovo cluster gratuito
3. Configura un utente database e le regole di accesso alla rete
4. Ottieni la stringa di connessione

**2.3 Configurazione del Server per la Connessione a MongoDB**

Crea un file .env nella directory server:

PORT=4000

MONGODB\_URI=mongodb://localhost:27017/blue-site-monitor

# Se utilizzi MongoDB Atlas, usa la stringa di connessione fornita

# MONGODB\_URI=mongodb+srv://<username>:<password>@<cluster>.mongodb.net/blue-site-monitor

JWT\_SECRET=your-secure-jwt-secret-key

NODE\_ENV=development

**3. Installazione delle Dipendenze**

Esegui il seguente comando nella directory principale per installare tutte le dipendenze:

npm run setup

Questo comando installerà:

* Le dipendenze root
* Le dipendenze del client
* Le dipendenze del server
* Le dipendenze dei server mock

**4. Configurazione dei Mock Server**

**4.1 Creazione del Mock Server BlueIOT**

Crea la struttura di directory:

mkdir -p mock-servers/blueiot-mock/src

Crea un file package.json nella directory mock-servers/blueiot-mock:

{

"name": "blueiot-mock-server",

"version": "1.0.0",

"scripts": {

"start": "ts-node src/server.ts"

},

"dependencies": {

"cors": "^2.8.5",

"express": "^4.18.2",

"ws": "^8.13.0"

},

"devDependencies": {

"@types/cors": "^2.8.13",

"@types/express": "^4.17.17",

"@types/ws": "^8.5.5",

"ts-node": "^10.9.1",

"typescript": "^5.1.3"

}

}

Crea il file src/server.ts:

*// mock-servers/blueiot-mock/src/server.ts*

import express from 'express';

import http from 'http';

import WebSocket from 'ws';

import cors from 'cors';

*// Configurazione*

const PORT = 4001;

const WS\_PORT = 4002;

*// Crea l'app Express per l'API REST*

const app = express();

app.use(cors());

app.use(express.json());

*// Rotte API*

app.get('/api/status', (req, res) => {

res.json({ status: 'OK', message: 'BlueIOT Mock Server is running' });

});

*// Avvia il server HTTP*

const httpServer = app.listen(PORT, () => {

console.log(`BlueIOT Mock REST API running on port ${PORT}`);

});

*// Crea il server WebSocket*

const wss = new WebSocket.Server({ port: WS\_PORT });

*// Variabili per la simulazione*

let connectedClients = new Set<WebSocket>();

let subscribedTags = new Set<string>();

let simulationInterval: NodeJS.Timeout | null = null;

*// Gestione delle connessioni WebSocket*

wss.on('connection', (ws) => {

console.log('Client connected to BlueIOT Mock WebSocket');

connectedClients.add(ws);

*// Invia risposta di connessione*

ws.send(JSON.stringify({

type: 'connection',

status: 'connected',

timestamp: new Date().toISOString()

}));

*// Gestione dei messaggi in arrivo*

ws.on('message', (message) => {

try {

const data = JSON.parse(message.toString());

console.log('Received message:', data);

*// Gestione diversi tipi di messaggi*

switch (data.type) {

case 'auth':

handleAuth(ws, data);

break;

case 'subscribe':

handleSubscribe(ws, data);

break;

case 'unsubscribe':

handleUnsubscribe(ws, data);

break;

default:

console.log(`Unhandled message type: ${data.type}`);

}

} catch (err) {

console.error('Error parsing message:', err);

}

});

*// Gestione della disconnessione*

ws.on('close', () => {

console.log('Client disconnected from BlueIOT Mock WebSocket');

connectedClients.delete(ws);

*// Se non ci sono più client connessi, ferma la simulazione*

if (connectedClients.size === 0 && simulationInterval) {

clearInterval(simulationInterval);

simulationInterval = null;

}

});

*// Avvia la simulazione se non è già in corso*

if (!simulationInterval) {

startSimulation();

}

});

*// Funzioni di gestione messaggi*

function handleAuth(ws: WebSocket, data: any) {

console.log(`Authentication request from ${data.username}`);

*// Simulazione di risposta di autenticazione (sempre riuscita)*

ws.send(JSON.stringify({

type: 'auth\_response',

success: true,

timestamp: new Date().toISOString()

}));

}

function handleSubscribe(ws: WebSocket, data: any) {

if (!data.tagIds || !Array.isArray(data.tagIds)) {

console.error('Invalid subscribe request: tagIds missing or not an array');

return;

}

console.log(`Subscribing to tags: ${data.tagIds.join(', ')}`);

*// Aggiungi i tag all'insieme degli iscritti*

data.tagIds.forEach((tagId: string) => subscribedTags.add(tagId));

*// Invia conferma*

ws.send(JSON.stringify({

type: 'subscription\_response',

success: true,

tagIds: data.tagIds,

timestamp: new Date().toISOString()

}));

}

function handleUnsubscribe(ws: WebSocket, data: any) {

if (!data.tagIds || !Array.isArray(data.tagIds)) {

console.error('Invalid unsubscribe request: tagIds missing or not an array');

return;

}

console.log(`Unsubscribing from tags: ${data.tagIds.join(', ')}`);

*// Rimuovi i tag dall'insieme degli iscritti*

data.tagIds.forEach((tagId: string) => subscribedTags.delete(tagId));

*// Invia conferma*

ws.send(JSON.stringify({

type: 'unsubscription\_response',

success: true,

tagIds: data.tagIds,

timestamp: new Date().toISOString()

}));

}

*// Simulazione di dati*

function startSimulation() {

const defaultTags = ['TAG001', 'TAG002', 'TAG003', 'TAG004', 'TAG005', 'TAG101', 'TAG102', 'TAG103', 'TAG104'];

*// Aggiungi i tag di default all'insieme*

defaultTags.forEach(tag => subscribedTags.add(tag));

*// Genera posizioni casuali per ogni tag ogni 2 secondi*

simulationInterval = setInterval(() => {

if (connectedClients.size === 0) return;

const positions = Array.from(subscribedTags).map(tagId => {

*// Genera coordinate casuali*

*// Basate sulla stessa posizione generale, con piccole variazioni*

const baseX = tagId.startsWith('TAG10') ? 250 : 150;

const baseY = tagId.startsWith('TAG10') ? 180 : 120;

return {

tagId,

x: baseX + (Math.random() - 0.5) \* 20,

y: baseY + (Math.random() - 0.5) \* 20,

z: 0,

mapId: 'map1',

batteryLevel: Math.round(70 + Math.random() \* 30),

timestamp: new Date().toISOString()

};

});

*// Invia l'aggiornamento di posizione a tutti i client connessi*

const positionUpdate = {

type: 'position',

data: positions,

timestamp: new Date().toISOString()

};

broadcastToAll(positionUpdate);

*// Occasionalmente, invia aggiornamenti della batteria*

if (Math.random() < 0.2) {

const batteryUpdates = generateBatteryUpdates();

if (batteryUpdates.length > 0) {

broadcastToAll({

type: 'battery',

data: batteryUpdates,

timestamp: new Date().toISOString()

});

}

}

*// Molto raramente, genera un allarme*

if (Math.random() < 0.05) {

generateAlarm();

}

}, 2000);

}

function generateBatteryUpdates() {

const updates = [];

*// Seleziona casualmente alcuni tag per aggiornamenti batteria*

for (const tagId of subscribedTags) {

if (Math.random() < 0.3) {

updates.push({

tagId,

level: Math.round(20 + Math.random() \* 80),

timestamp: new Date().toISOString()

});

}

}

return updates;

}

function generateAlarm() {

if (subscribedTags.size === 0) return;

*// Scegli un tag casuale*

const tagIds = Array.from(subscribedTags);

const randomTagId = tagIds[Math.floor(Math.random() \* tagIds.length)];

*// Tipi di allarme possibili*

const alarmTypes = ['geofence', 'emergency', 'lowBattery', 'tagOffline'];

const randomType = alarmTypes[Math.floor(Math.random() \* alarmTypes.length)];

const alarm = {

type: 'alarm',

data: {

alarmType: randomType,

tagId: randomTagId,

timestamp: new Date().toISOString(),

message: `${randomType.charAt(0).toUpperCase() + randomType.slice(1)} alarm triggered for tag ${randomTagId}`

}

};

broadcastToAll(alarm);

}

function broadcastToAll(data: any) {

const message = JSON.stringify(data);

connectedClients.forEach(client => {

if (client.readyState === WebSocket.OPEN) {

client.send(message);

}

});

}

console.log(`BlueIOT Mock WebSocket server running on port ${WS\_PORT}`);

**4.2 Creazione del Mock Server CRM**

Crea la struttura di directory:

mkdir -p mock-servers/crm-mock/src

Crea un file package.json nella directory mock-servers/crm-mock:

{

"name": "crm-mock-server",

"version": "1.0.0",

"scripts": {

"start": "ts-node src/server.ts"

},

"dependencies": {

"cors": "^2.8.5",

"express": "^4.18.2"

},

"devDependencies": {

"@types/cors": "^2.8.13",

"@types/express": "^4.17.17",

"ts-node": "^10.9.1",

"typescript": "^5.1.3"

}

}

Crea il file src/server.ts:

*// mock-servers/crm-mock/src/server.ts*

import express from 'express';

import cors from 'cors';

*// Configurazione*

const PORT = 4002;

*// Crea l'app Express*

const app = express();

app.use(cors());

app.use(express.json());

*// Dati simulati*

const companies = [

{

id: 'company1',

name: 'Construction Corp Inc.',

address: '123 Builder Street, New York, NY',

contact: 'John Builder',

phone: '+1-234-567-8901',

email: 'info@constructioncorp.com'

}

];

const employees = [

{

id: '1',

companyId: 'company1',

name: 'John Doe',

email: 'john.doe@example.com',

phone: '+1234567890',

position: 'Construction Worker',

department: 'Operations',

siteId: 'site1'

},

{

id: '2',

companyId: 'company1',

name: 'Jane Smith',

email: 'jane.smith@example.com',

phone: '+1234567891',

position: 'Site Manager',

department: 'Management',

siteId: 'site1'

},

{

id: '3',

companyId: 'company1',

name: 'Robert Johnson',

email: 'robert.johnson@example.com',

phone: '+1234567892',

position: 'Electrician',

department: 'Technical',

siteId: 'site1'

},

{

id: '4',

companyId: 'company1',

name: 'Maria Garcia',

email: 'maria.garcia@example.com',

phone: '+1234567893',

position: 'Safety Officer',

department: 'Safety',

siteId: 'site2'

},

{

id: '5',

companyId: 'company1',

name: 'David Kim',

email: 'david.kim@example.com',

phone: '+1234567894',

position: 'Architect',

department: 'Design',

siteId: 'site2'

}

];

const assets = [

{

id: '101',

companyId: 'company1',

name: 'Excavator #1',

type: 'Heavy Equipment',

serialNumber: 'EX-2023-001',

description: 'Caterpillar 320 Excavator',

siteId: 'site1'

},

{

id: '102',

companyId: 'company1',

name: 'Concrete Mixer',

type: 'Heavy Equipment',

serialNumber: 'CM-2023-002',

description: 'Mobile concrete mixer truck',

siteId: 'site1'

},

{

id: '103',

companyId: 'company1',

name: 'Power Generator',

type: 'Equipment',

serialNumber: 'PG-2023-003',

description: 'Portable power generator 5000W',

siteId: 'site1'

},

{

id: '104',

companyId: 'company1',

name: 'Toolbox A',

type: 'Tool',

serialNumber: 'TB-2023-004',

description: 'Complete set of hand tools',

siteId: 'site1'

},

{

id: '105',

companyId: 'company1',

name: 'Safety Equipment Kit',

type: 'Safety Equipment',

serialNumber: 'SE-2023-005',

description: 'Safety helmets, vests and gear',

siteId: 'site2'

},

{

id: '106',

companyId: 'company1',

name: 'Crane #2',

type: 'Heavy Equipment',

serialNumber: 'CR-2023-006',

description: 'Mobile crane for heavy lifting',

siteId: 'site2'

}

];

*// Rotte API*

*// Autenticazione*

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

const { username, password } = req.body;

if (!username || !password) {

return res.status(400).json({ message: 'Username and password are required' });

}

*// In un'implementazione mock, accettiamo qualsiasi credenziale valida*

const token = `mock-token-${Date.now()}`;

res.json({

token,

user: {

id: 'user1',

username,

email: `${username}@example.com`,

firstName: 'John',

lastName: 'Doe',

role: 'admin',

companyId: 'company1'

}

});

});

*// Informazioni azienda*

app.get('/companies/:companyId', (req, res) => {

const { companyId } = req.params;

const company = companies.find(c => c.id === companyId);

if (!company) {

return res.status(404).json({ message: 'Company not found' });

}

res.json(company);

});

*// Dipendenti azienda*

app.get('/companies/:companyId/employees', (req, res) => {

const { companyId } = req.params;

const { siteId } = req.query;

let result = employees.filter(e => e.companyId === companyId);

if (siteId) {

result = result.filter(e => e.siteId === siteId);

}

res.json(result);

});

*// Asset azienda*

app.get('/companies/:companyId/assets', (req, res) => {

const { companyId } = req.params;

const { siteId } = req.query;

let result = assets.filter(a => a.companyId === companyId);

if (siteId) {

result = result.filter(a => a.siteId === siteId);

}

res.json(result);

});

*// Rotta di stato*

app.get('/api/status', (req, res) => {

res.json({ status: 'OK', message: 'CRM Mock Server is running' });

});

*// Avvia il server*

app.listen(PORT, () => {

console.log(`CRM Mock Server running on port ${PORT}`);

});

**5. Avvio dell'Applicazione**

**5.1 Avvio in Modalità Sviluppo**

Per avviare l'intera applicazione in modalità sviluppo:

npm run dev

Questo comando avvierà:

* Il server backend su <http://localhost:4000>
* Il client frontend su <http://localhost:3000>
* Il mock server BlueIOT su <http://localhost:4001> (API REST) e ws://localhost:4002 (WebSocket)
* Il mock server CRM su <http://localhost:4002>

**5.2 Avvio dei Componenti Separatamente**

Se preferisci avviare i componenti separatamente:

*# Client (frontend)*

npm run client

*# Server (backend)*

npm run server

*# Mock server BlueIOT*

npm run mock-blueiot

*# Mock server CRM*

npm run mock-crm

**6. Build per la Produzione**

**6.1 Build dei Componenti**

Per creare una build di produzione:

npm run build

Questo comando creerà:

* Una build ottimizzata del frontend nella directory client/dist
* Una build compilata del backend nella directory server/dist

**6.2 Configurazione per Produzione**

Per la produzione, crea un file .env nella directory server con le seguenti impostazioni:

PORT=4000

MONGODB\_URI=mongodb://your-production-mongodb-uri

JWT\_SECRET=your-secure-production-jwt-secret

NODE\_ENV=production

**6.3 Avvio in Produzione**

Per avviare l'applicazione in produzione:

*# Naviga nella directory server*

cd server

*# Avvia il server*

npm start

**7. Configurazione Nginx per Produzione (Opzionale)**

Se desideri utilizzare Nginx come proxy inverso, segui questi passaggi:

**7.1 Installa Nginx**

**Ubuntu:**

sudo apt update

sudo apt install nginx

**macOS:**

brew install nginx

**7.2 Configura Nginx**

Crea un file di configurazione per il tuo sito:

*# Ubuntu*

sudo nano /etc/nginx/sites-available/blue-site-monitor

*# macOS*

nano /usr/local/etc/nginx/servers/blue-site-monitor.conf

Aggiungi la seguente configurazione:

server {

listen 80;

server\_name your-domain.com; *# Cambia con il tuo dominio o IP*

location / {

*# Servi i file statici del frontend*

root /path/to/blue-site-monitor/client/dist;

try\_files $uri $uri/ /index.html;

}

location /api {

*# Proxy verso il server backend*

proxy\_pass http://localhost:4000;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

}

location /ws {

*# Proxy verso il server WebSocket*

proxy\_pass http://localhost:4002;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection "Upgrade";

proxy\_set\_header Host $host;

}

}

**Attiva la configurazione (solo Ubuntu):**

sudo ln -s /etc/nginx/sites-available/blue-site-monitor /etc/nginx/sites-enabled/

sudo nginx -t *# Verifica la configurazione*

sudo systemctl restart nginx

**Riavvia Nginx (macOS):**

brew services restart nginx

**8. Risoluzione dei Problemi Comuni**

**8.1 Problemi di Connessione a MongoDB**

**Problema:** Impossibile connettersi a MongoDB **Soluzione:**

* Verifica che MongoDB sia in esecuzione: sudo systemctl status mongod (Linux) o brew services list (macOS)
* Controlla l'URI di connessione nel file .env
* Verifica le regole firewall o le regole di accesso alla rete per MongoDB Atlas

**8.2 Problemi con le Dipendenze**

**Problema:** Errori durante l'installazione delle dipendenze **Soluzione:**

* Cancella la cartella node\_modules e il file package-lock.json in ciascuna directory
* Esegui nuovamente npm run setup
* Verifica la versione di Node.js: node -v (dovrebbe essere 16+)

**8.3 Problemi con il Client**

**Problema:** Il client non si avvia o non si connette al backend **Soluzione:**

* Verifica che le porte configurate in vite.config.ts siano corrette
* Controlla che il server backend sia in esecuzione
* Verifica i log del client per errori specifici

**8.4 Problemi con i Mock Server**

**Problema:** I mock server non funzionano correttamente **Soluzione:**

* Verifica che non ci siano conflitti di porta
* Controlla i log dei mock server per errori specifici
* Riavvia i mock server

**9. Utilizzo dell'Applicazione**

Una volta avviata l'applicazione, puoi accedervi nel tuo browser all'indirizzo <http://localhost:3000>

**9.1 Login**

Usa queste credenziali per accedere:

* Username: admin
* Password: password

(In modalità sviluppo con mock server, qualsiasi combinazione funzionerà)

**9.2 Dashboard**

Dopo il login, verrai indirizzato alla dashboard che mostra una panoramica del sistema.

**9.3 Funzionalità Principali**

* **Dashboard**: Visualizza statistiche e notifiche
* **Map View**: Visualizza la posizione in tempo reale di dipendenti e asset
* **Employees**: Gestisci i dipendenti e assegna loro i tag
* **Assets**: Gestisci gli asset e assegna loro i tag
* **Configuration**: Configura mappe, antenne e geofence (solo admin)
* **Settings**: Configura le impostazioni dell'applicazione

**10. Estensione e Personalizzazione**

**10.1 Integrazione con un Sistema di Posizionamento Reale**

Per integrare con un sistema di posizionamento reale come BlueIOT:

1. Modifica client/src/services/positioning-service.ts per aggiornare i parametri di connessione
2. Aggiorna client/src/services/positioning/blueiot-provider.ts per supportare il protocollo specifico

**10.2 Integrazione con un CRM Reale**

Per integrare con un CRM reale:

1. Modifica client/src/services/crm-service.ts per aggiornare i parametri di connessione
2. Aggiorna client/src/services/crm/default-provider.ts per implementare le chiamate API specifiche

**10.3 Estensione con Nuove Funzionalità**

Per aggiungere nuove funzionalità:

1. Aggiungi componenti Vue nella directory client/src/components/
2. Aggiungi nuove viste nella directory client/src/views/
3. Aggiorna client/src/router/index.ts per aggiungere nuove rotte
4. Implementa nuovi store Pinia se necessario in client/src/stores/
5. Aggiungi nuove API nel backend in server/src/routes/

**Conclusione**

Questa guida completa ti ha fornito tutti i passaggi necessari per installare, configurare e avviare l'applicazione BlueIOT Site Monitor. Seguendo queste istruzioni, dovresti essere in grado di avere un'applicazione funzionante in modalità sviluppo, pronta per essere adattata alle tue esigenze specifiche.

La modalità di sviluppo con i mock server ti consente di testare e sviluppare l'applicazione senza dipendere dai sistemi reali, mentre l'architettura modulare ti permette di sostituire facilmente i provider mock con implementazioni reali quando sei pronto per la produzione.

**Configurazione Apache per BlueIOT Site Monitor**

Se preferisci utilizzare Apache invece di Nginx come server web per la produzione, ecco una guida completa su come configurarlo. Apache è un'alternativa valida e molto diffusa a Nginx, particolarmente comune in ambienti di hosting condiviso.

**1. Installazione di Apache**

**Per Ubuntu/Debian:**

bash

sudo apt update

sudo apt install apache2

**Per macOS:**

bash

brew install httpd

**Per Windows:**

Scaricare e installare XAMPP (include Apache) da <https://www.apachefriends.org/>

**2. Abilitazione dei Moduli Necessari**

Per Ubuntu/Debian, abilita i moduli necessari per proxy e WebSocket:

bash

sudo a2enmod proxy

sudo a2enmod proxy\_http

sudo a2enmod proxy\_wstunnel

sudo a2enmod rewrite

**3. Configurazione del Virtual Host**

**3.1 Ubuntu/Debian:**

Crea un nuovo file di configurazione:

bash

sudo nano /etc/apache2/sites-available/blue-site-monitor.conf

**3.2 macOS (con Homebrew):**

bash

nano /usr/local/etc/httpd/extra/httpd-vhosts.conf

**3.3 Windows (con XAMPP):**

Modifica il file:

C:\xampp\apache\conf\extra\httpd-vhosts.conf

**4. Configurazione del Virtual Host**

Aggiungi la seguente configurazione al file, sostituendo your-domain.com con il tuo dominio o indirizzo IP:

apache

<VirtualHost \*:80>

ServerName your-domain.com

ServerAlias www.your-domain.com

# Documentroot punta alla directory dist del frontend

DocumentRoot /path/to/blue-site-monitor/client/dist

<Directory /path/to/blue-site-monitor/client/dist>

Options -Indexes +FollowSymLinks

AllowOverride All

Require all granted

# Rewrite per SPA (applicazioni a pagina singola)

RewriteEngine On

RewriteBase /

# Se il file richiesto non esiste, reindirizza a index.html

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteRule ^(.\*)$ index.html [L,QSA]

</Directory>

# Proxy per l'API backend

<Location /api>

ProxyPass http://localhost:4000

ProxyPassReverse http://localhost:4000

# Preserva l'host originale

ProxyPreserveHost On

</Location>

# Proxy per il WebSocket BlueIOT

<Location /ws>

ProxyPass ws://localhost:4002

ProxyPassReverse ws://localhost:4002

# Configurazione specifica per WebSocket

RewriteEngine On

RewriteCond %{HTTP:Upgrade} =websocket [NC]

RewriteCond %{HTTP:Connection} =Upgrade [NC]

RewriteRule ^/?(.\*) ws://localhost:4002/$1 [P,L]

</Location>

# Log files

ErrorLog ${APACHE\_LOG\_DIR}/blue-site-error.log

CustomLog ${APACHE\_LOG\_DIR}/blue-site-access.log combined

</VirtualHost>

**Note sulla configurazione:**

* DocumentRoot punta alla cartella dist del frontend compilato
* <Directory> imposta i permessi e le regole di rewrite per l'applicazione SPA
* <Location /api> configura il proxy per l'API backend
* <Location /ws> configura il proxy per i WebSocket di BlueIOT

**5. Attivazione della Configurazione**

**5.1 Ubuntu/Debian:**

bash

*# Attiva il sito*

sudo a2ensite blue-site-monitor.conf

*# Verifica la configurazione*

sudo apache2ctl configtest

*# Riavvia Apache*

sudo systemctl restart apache2

**5.2 macOS:**

bash

*# Riavvia Apache*

brew services restart httpd

**5.3 Windows (XAMPP):**

Riavvia Apache dal pannello di controllo di XAMPP.

**6. Configurazione HTTPS (Raccomandata per Produzione)**

**6.1 Ottenere un Certificato SSL con Let's Encrypt (Ubuntu/Debian)**

bash

*# Installa Certbot*

sudo apt install certbot python3-certbot-apache

*# Ottieni un certificato e configura automaticamente Apache*

sudo certbot --apache -d your-domain.com -d www.your-domain.com

**6.2 Configurazione Manuale di SSL**

Se non utilizzi Let's Encrypt, aggiungi queste righe al tuo Virtual Host per SSL:

apache

<VirtualHost \*:443>

ServerName your-domain.com

ServerAlias www.your-domain.com

# Configurazione SSL

SSLEngine on

SSLCertificateFile /path/to/your/certificate.crt

SSLCertificateKeyFile /path/to/your/private.key

# Se hai un certificato chain

SSLCertificateChainFile /path/to/your/chain.crt

# Stessa configurazione di prima...

DocumentRoot /path/to/blue-site-monitor/client/dist

# ... (resto della configurazione come sopra)

</VirtualHost>

**7. SELinux (Solo per CentOS/RHEL/Fedora)**

Se utilizzi una distribuzione con SELinux attivo, dovrai configurare i permessi appropriati:

bash

*# Consenti ad Apache di effettuare connessioni di rete*

sudo setsebool -P httpd\_can\_network\_connect 1

*# Se memorizzi file in una directory non standard*

sudo chcon -R -t httpd\_sys\_content\_t /path/to/blue-site-monitor/client/dist

**8. Configurazione Apache come Servizio**

**8.1 Esecuzione Automatica all'Avvio su Ubuntu/Debian:**

L'installazione standard configura già Apache come servizio. Per verificare:

bash

sudo systemctl status apache2

**8.2 Esecuzione Automatica all'Avvio su macOS:**

bash

brew services start httpd

**8.3 Esecuzione Automatica all'Avvio su Windows:**

Configurare XAMPP per l'avvio automatico tramite le impostazioni di Windows.

**9. Ottimizzazioni per la Produzione**

**9.1 Compressione dei Contenuti**

Abilita la compressione gzip/deflate:

apache

<IfModule mod\_deflate.c>

AddOutputFilterByType DEFLATE text/html text/plain text/xml text/css text/javascript application/javascript application/json

</IfModule>

**9.2 Caching**

Attiva il caching per migliorare le prestazioni:

apache

<IfModule mod\_expires.c>

ExpiresActive On

ExpiresByType image/jpg "access plus 1 year"

ExpiresByType image/jpeg "access plus 1 year"

ExpiresByType image/gif "access plus 1 year"

ExpiresByType image/png "access plus 1 year"

ExpiresByType image/svg+xml "access plus 1 year"

ExpiresByType text/css "access plus 1 month"

ExpiresByType text/javascript "access plus 1 month"

ExpiresByType application/javascript "access plus 1 month"

ExpiresByType application/json "access plus 0 seconds"

</IfModule>

**9.3 Sicurezza**

Aggiungi header di sicurezza:

apache

<IfModule mod\_headers.c>

Header set X-Content-Type-Options "nosniff"

Header set X-XSS-Protection "1; mode=block"

Header set X-Frame-Options "SAMEORIGIN"

Header set Strict-Transport-Security "max-age=31536000; includeSubDomains"

</IfModule>

**10. Gestione dell'Applicazione Node.js Backend**

**10.1 Utilizzo di PM2 per Gestire il Backend**

Per mantenere il backend Node.js in esecuzione in modo affidabile:

bash

*# Installa PM2 globalmente*

npm install -g pm2

*# Avvia l'applicazione backend con PM2*

cd /path/to/blue-site-monitor/server

pm2 start dist/app.js --name "blue-site-backend"

*# Configura PM2 per avviarsi al boot*

pm2 startup

pm2 save

**10.2 Configurazione dei Log PM2**

bash

*# Visualizza i log in tempo reale*

pm2 logs blue-site-backend

*# Configura rotazione dei log*

pm2 install pm2-logrotate

**Conclusione**

La configurazione di Apache è una valida alternativa a Nginx per il tuo progetto BlueIOT Site Monitor. Funziona particolarmente bene in ambienti dove Apache è già installato o preferito per politiche aziendali.

La configurazione sopra illustrata:

1. Serve i file statici del frontend compilato
2. Gestisce il routing per applicazioni SPA
3. Forwarda le richieste API al backend Node.js
4. Supporta WebSocket per comunicazioni in tempo reale
5. Include ottimizzazioni per la produzione e sicurezza

Una volta completata la configurazione di Apache, l'applicazione sarà accessibile all'indirizzo del dominio che hai specificato, con tutte le funzionalità backend e WebSocket funzionanti correttamente.