### High-Level Process for Sprint 1:

**1. Backend Infrastructure Setup:**

* Initialize a new Node.js project using npm or yarn.
* Integrate Express.js as the backend framework.
* Install necessary dependencies such as body-parser, mongoose (if using MongoDB), pg (if using PostgreSQL), jsonwebtoken, bcryptjs for password hashing, and others you might require.

**2. Database Schema Design & Setup:**

* Choose between MongoDB or PostgreSQL based on your project’s needs (For this example, let’s consider MongoDB).
* Design the user schema. A basic schema might have fields like username, hashedPassword, email, created\_at, etc.
* Design the device schema, which might include fields like device\_uuid, name, added\_at, thresholds, etc.

**3. API Endpoints Development:**

* **User Sign-up Endpoint**:
  + Validate incoming data (e.g., check email format, password strength).
  + Hash the password using bcrypt.
  + Save the user to the database.
  + Return a success response or appropriate error messages.
* **User Login Endpoint**:
  + Validate the user’s credentials.
  + If valid, generate a JWT (JSON Web Token) and send it in the response.
  + Store this token on the client side for subsequent authenticated requests.
* **Add Device Endpoint**:
  + Validate incoming device data.
  + Store the new device in the database with its related user.
  + Return the device data or appropriate error messages.
* **List Devices Endpoint**:
  + Fetch all devices related to the authenticated user from the database.
  + Return the list of devices.

**4. MQTT Integration**:

* Integrate Mosquitto MQTT broker to handle data subscription.
* Establish an MQTT connection when the backend server starts.
* Subscribe to the topic 'things/+/shadow/update' to get updates from the Thingy device.

### Architectural Design:

1. **Backend (Express.js)**
   * This will house our business logic, API endpoints, and MQTT connection.
   * It will communicate with the database for data persistence.
2. **Database (MongoDB or PostgreSQL)**
   * For storing user data, device data, and other necessary configurations.
3. **Docker**
   * Use Docker to containerize the Express backend and the chosen database.
   * Create a Dockerfile for the backend, detailing the environment, dependencies, and the run command.
   * Use docker-compose to orchestrate multi-container Docker applications. Your docker-compose.yml should define services for the backend and the database. These services should be networked together so they can communicate.
4. **GitLab CI (.gitlab-ci.yml file)**
   * Define the pipeline stages, which might be build, test, and maybe deploy.
   * Under the test stage, run unit tests for your backend logic. For Node.js, you can use frameworks like Mocha or Jest for testing.
   * If any test fails, the pipeline should also fail, ensuring that no faulty code gets deployed.

**Example .gitlab-ci.yml Configuration**:

image: node:latest  
  
stages:  
 - build  
 - test  
  
cache:  
 paths:  
 - node\_modules/  
  
before\_script:  
 - npm install  
  
build:  
 stage: build  
 script:  
 - npm run build  
  
test:  
 stage: test  
 script:  
 - npm run test