Multi-tier IoT System simulation

Nirbhay Singh, Tanmay Khandelwal

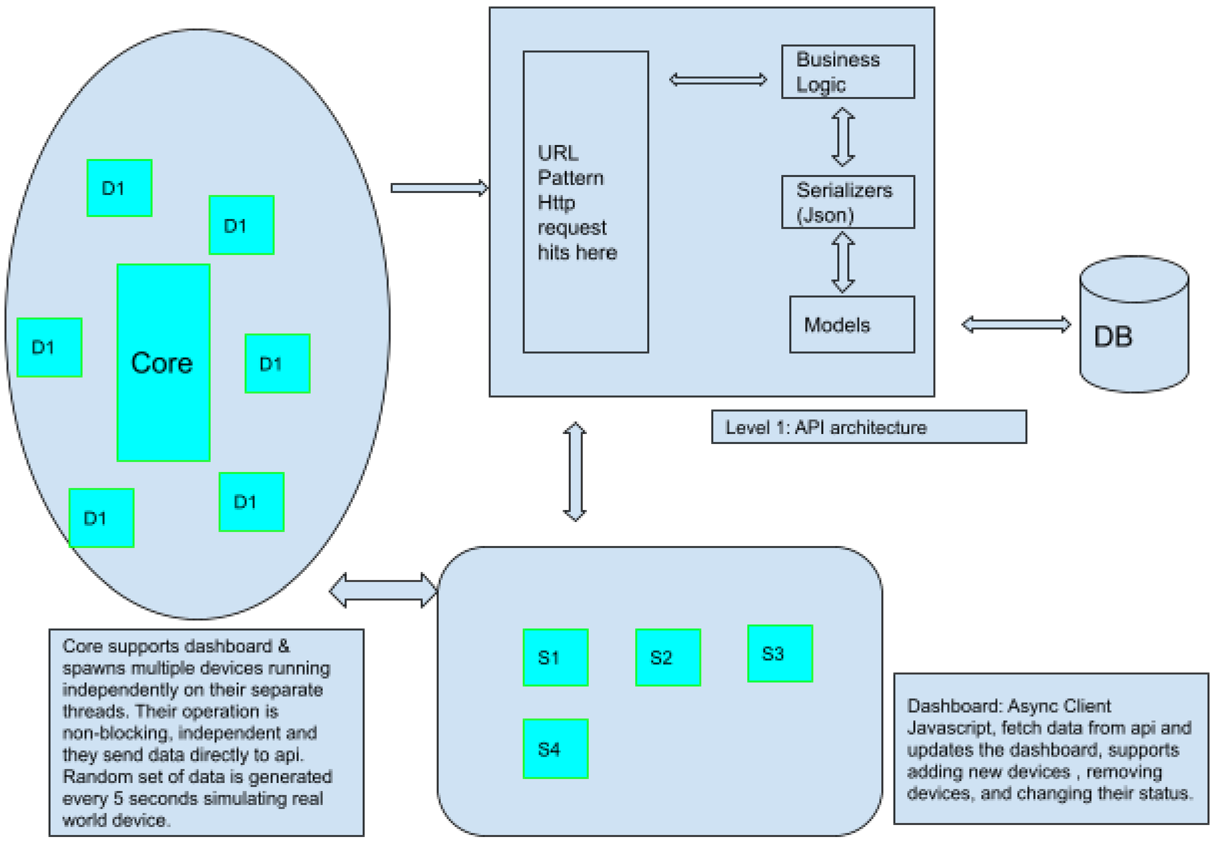
18ucc029@lnmiit.ac.in, 18ucc145@lnmiit.ac.in

Abstract

Internet of things, a buzzword only a few years ago is now embedded in our lives and is on a trajectory to take even greater part in our lives with advent of low-latency networks and falling cost of hardware. Given its potential for very wide applicability to almost all verticals and aspects of business, industries, manufacturing, consumer goods, supply chains etc..IoT as a whole is a very broad area.This paper explores the development of a multi-tier system, consisting of devices and their interfaces for other devices and users both. It can have its application in a shared work environment or even in Homes. Devices are simulated to run independently and the project demonstrates the capabilities of the system which can be suited to needs of the user.

1.1 Introduction

The project consists of three main subsystems: the Core, rest API and Dashboard which also contains an asynchronous javascript client which accesses the API and fetches the data at regular intervals published by independent devices running in Core on separate threads. Core also acts as a backend for Dashboard connecting using a persistent websocket connection. Api persists the published data on a database. Further in the paper we discuss each of the subsystems in detail including their workflow , testing and how they integrate with each other. Multithreaded nature of Core and asynchronous operations of javascript the client would be discussed. Internal nature and flows of API is discussed further.


D1
D1
Core
D1
D1
D1
D1
D1
URL Pattern Http request hits here
Business Logic
Serializers (Json)
Models
DB
Level 1: API architecture
S1
S2
S3
S4
Core supports dashboard & spawns multiple devices running independently on their separate threads. Their operation is non-blocking, independent and they send data directly to api. Random set of data is generated every 5 seconds simulating real world device.
Dashboard: Async Client Javascript, fetch data from api and updates the dashboard, supports adding new devices , removing devices, and changing their status.

1.2 Core

Technical and Functional specification of core:

- Written in Java 11 openjdk, and utilizes websocket protocol to connect to dashboard.
- Spawns multiple independent devices running on separate threads.
- Devices connect to API and perform POST and PUT operations to push and update data of each device identified uniquely by their Identifier.
- Data is shared with the API as a Json string containing ID and randomly generated data by each device, simulating real-world behaviour.
- Other than managing Devices, the main thread also connects with the dashboard using websocket messages OnOpen(), OnClose(), OnMessage() and OnError().
- To connect with the API, http requests and responses are used.
- Each device sends randomly generated data periodically to the API
- Device class is made a Runnable which implies that it can be run on separate thread simulating independence.

1.3 The API

Technical and functional specifications of API:

- Written in Python using django rest-framework.
- Development server intercepts the http requests received from different devices and dashboard.
- Http request is matched with URL patterns, and farther passed to appropriate views (business logic).
- Views parse the request and use serializers to convert them into Json and from json to http response.
- Models are central to all the logic and define the schema of the database.
- Based on models serializers are defined.
- Since django has its own ORM (object relational mapper) for database access, queryset is obtained for model Device as Device.objects.all()
- API is tested on Postman.

Endpoints defined and supported:

Endpoint	Method
/api/devices	GET, POST, DELETE
/api/devices/<int:pk>	PUT, GET, DELETE; pk is used as ID of devices

1.4 DashBoard Client:

Third part of the system is the client hosted on core. It creates tiles for each device and connects with Core on websocket connection. It also connects with API asynchronously and fetches updates in realtime. Those updates are then updated on the tiles in real-time in non-blocking fashion.

Fetch API provided by javascript is used to get updates from the API, function api_access() is run at an interval of 6 seconds to get updates. It starts when the first device is added to the pool and stops when the server closes or there is interruption from the websocket connection that is OnClose().

Multiple instances of Dashboard client can be created and each instance would receive the same real-time update. This kind of system finds its utility in a shared work environment where multiple stakeholders need updates simultaneously not only from devices but from each other.

1.5 Conclusion

We demonstrated the technological feasibility of an IOT system capable of hosting multitudes of devices and connecting and sharing updates in real time with users and other stakeholders. We used a websocket connection and rest API to connect various subsystems. We also simulated behaviour of real-life devices running on independent threads and communicating with the outside world on http connection. Technology stack used is as follows:

Technology	Comment
Java 11 open JDK Jboss development Server Java Multithreading: Runnable and Threadpool	/java/net/http to make http requests; javax websocket implementation : to connect to dashboard /javax/websocket each device was run on separate thread creating a non-blocking access to the api
Python3.6 Django 3.2.* Django_rest_framework Sqlite database Postman for API Testing	To create REST Api and to persist data from each device uniquely identified by its ID
Html, CSS, Vanilla Javascript	To create the dashboard, fetch api is used to fetch data from the API in a non-blocking manner.

Acknowledgements: We would like to thank the Open Source software community for maintaining high quality software and documentation and the wonderful community of maintainers of the projects like Django. We would also like to thank Dr. Abhishek Sharma for inspiring us over the course of this project.

Bibliography

[1] OpenJdk documentation of java http implementation.
[2] Mozilla docs for Asynchronous javascript and related technologies.
[3] Django documentation version 3.0.*