FOSS IoT Frameworks review

Benoit RENAULT

2017/08/10

Contents

1 Introduction **3**

2 Evaluation criteria **3**
2.1 General criteria . . . 3
2.2 IoT-specific criteria . . . 4
2.3 Application-specific criteria . . . 4

3 Detailed analysis **5**
3.1 SiteWhere . . . 5
3.2 Kaa . . . 7
3.3 Device Hive . . . 9
3.4 Zetta JS . . . 12
3.5 Parse . . . 14
3.6 DSA . . . 15
3.7 Blynk . . . 16
3.8 Paho . . . 17
3.9 Kura . . . 18
3.10 Kapua . . . 19
3.11 Hono . . . 20

4 Summarized analysis **21**

5 Conclusion **21**

1 Introduction

The context for this document is the need for an IoT middleware platform for building, managing, and integrating connected products for the OCCIware LinkedData Use Case. As the OCCIware project is FOSS[1], the component **must** be open-source itself. Therefore, what you will find here is a short state of the art of such frameworks, and no consideration shall be given to proprietary solutions.

Please note that, while the evaluation criteria may be considered timeless, the analyses presented below only reflect the state of the different projects as of this document's last update date, written on the first page.

2 Evaluation criteria

Each of the criteria is presented below with at least one argument as to how it allows to measure a given characteristic. The criteria are also chosen so that each solution needs not be analyzed more than two hours. For qualitative criteria, a little argumentation will be given in the detailed analysis (noted with excellent/good/average/poor/non-existent). Rational critics are, of course, welcome.

Some of them come from , and the ones about security have been inspired by Craig SMITH's "IoT Framework Assessment" OWASP page.

2.1 General criteria

Here are some general criteria that allow us to assess the quality of any FOSS project.

Project Health Determines how "mature" the project is, and its potential to last on the long term.

- First Release Date: The older, the better, might clue a position of pionneer, or a certain stability.
- Latest Release Date: The newer, the better, might clue that the project's release cycle is still active.
- Latest Commit Date: The newer, the better, if the project's release cycle is slow, might clue that the project is still ongoing active development.
- Number of main contributors: The higher, the better. If equal to one, exercise caution. Main contributors have proportional contributions to the code repository.
- Open issues ratio: The lower, the better. Is equal to the number of open issues over the total number of issues. Might clue at a good responsivity to user input, be it for bugs or improvements.

Company backing Partly shows how reliable the software can be: if there is a company ready to back it up and provide support to customers, and if there are other companies using the software, it might mean that it is trustworthy.

- Company Support: The better the company support is, the better the appreciation.
- Company Adoption: The more the software is adopted by other actors, the better the appreciation.

Documentation A well-documented project is a must.

- Currentness: The documentation must be up to date with the code. The more it respects this principle, the better the appreciation.
- Adaptability: Documentation must be adapted to the different users. The more it respects this principle, the better the appreciation.

[1] Free or Open-Source Software

UI-UX The software must provide appropriate tools at all levels, be it CLIs or Graphical UIs, depending on the context.

- Server Management UI: the more powerful and concise, the better the appreciation.
- Sample applications: the more there are and the more complete, the better the appreciation.

2.2 IoT-specific criteria

Here are some general criteria that allow us to assess the quality of IoT projects in particular.

Security A key characteristic for IoT systems. As we cannot conduct a security assessment ourselves, we will rather look for a commitment to security and whether audits have been made or not. Security must be ensured at all levels: connectivity, storage, update, authentication, appropriate management of devices...

- Statements: the more is explained on the security measures taken in the documentation, the better the appreciation. Clues at a concern and understanding of the IoT problematics.
- Audits: the more positive reviews about the security are, the better the appreciation. Clues at a good applications of best security practices.

Connectivity/Flexibility Is very important for IoT systems, because they are supposed to connect many elements that are all very different.

- Number of compatible hardware platforms: The higher, the better. Might clue to a shorter time-to-market.
- Number of supported protocols: The higher, the better. Might clue at a will not to lock the user in the system.
- Number of SDK implementations: The higher, the better. Might clue at an easier time for in-house developpers to incorporate the solution with their own prefered language.
- Modularity: the more the components of the solution can easily be changed (for instance, the DB), the better the appreciation.

Scalability As IoT systems tend to grow really fast, it is necessary that the application scales by design, and allows for many devices, all over the world, to connect to it, without failure.

- Third-Party scalability: IoT solutions are often a combination of many pre-existing FOSS projects. This criteria evaluates whether the 3rd-Party solutions chosen by the IoT platform are capable of scaling: the more they are, the better the appreciation.
- Platform scalability: IoT solutions must also be scalable themselves by being capable of being distributed across different machines and still be able to talk to one another smoothly. This criteria evaluates whether the platform is capable of scaling: the more it id, the better the appreciation.

2.3 Application-specific criteria

Here are some specific criteria that are related to the peculiar needs of the OCCIware project.

Arduino/NodeMCU compatibility Since it is the most popular platform for hardware experimentation, it is an important criteria that the service musts provide good support for it.

- Library: The more complete and recognized the library, the better the appreciation.
- Boilerplate code: The smaller the boilerplate code, the better the appreciation.

Java The framework must be based on Java for easy editing of the sources and better maintainability, since it is the main language of the author.

- Server percentage: The higher the quantity of Java code for the server, the better.

3 Detailed analysis

3.1 SiteWhere

Project Health

- First Release Date: The development started in 2010 as a closed source, asset tracking platform. The first open source release happened on **2014/03/06**. With 7 years of age, it can be assumed it is quite mature and stable.
- Latest Release Date: **2017/06/19** - Still releases new versions as of today. The project has an irregular release cycle, "When it's ready"-Debian-like.
- Latest Commit Date: **2017/06/20** - Work is still ongoing to deliver a new version.
- Number of main contributors: **1** - It seems to be mainly a one-man project, which is a problem. Some other people have contributed little bits of code, but not to the extent the main contributor has.
- Open issues ratio: **0.0534653465** - With 505 issues opened by many various actors (and not just the main contributor as milestones) since the project's arrival on Github, only 27 remain today, which is a very, very good. Excellent reactivity.

Good project health overall, though a shame that it seems to have only one main developer behind it.

Company backing

- Company Support: **excellent** - the project is backed by a well-established company (seven years of existence), and they offer a separate enterprise version with more features. Support is available to companies through the possibility to buy block hours for assistance, and also to anyone using the community edition through a rather active Google Group instance.
- Company Adoption: **good** - the corporate page of the project gives 2 comments by supposed customers, no success story using sitewhere was found but there were a few press articles talking about it like this one. It can however be infered that if the company has existed for 7 years, it means that they have found clients that appreciate their services.

Good company backing overall, though it's a shame they don't communicate a bit more about their customers.

Documentation

- Currentness: **good** - From the short time spent in the documentation, it seems it has been updated to match the latest release (as its number is written on the documentation homepage), however, in the absence of dating on all pages, it makes it more difficult to say if everything has been updated or not : for that, you have to go to the documentation Github Repository.
- Adaptability: **excellent** - The documentation is well separated between user and developper use cases. Explanations are clear, with lots of screenshots and appropriate commands where needed. The README.md on the Github Repository is crystal clear. Different levels of documentation are provided, like architecture, technologies, usage, code structure, ...

Good documentation overall, does make you want to use the product.

UI-UX

- Server Management UI: **good** - Powerful and well-documented web administration interface, though its design is a little bit old school.
- Sample applications: **good** - There are a few example applications for ios, android and webapps but not so many examples for hardware.

The proposed UIs seem good overall, but it would take a deeper examination to fully conclude on their capacity.

Security

- Statements: **average** - There is no open commitment to a secure system, on none of their websites. No explanation of the security measures they have taken, except that they use the Spring Frameworks integrated capabilities (which is good). No mention about the cryptographic means they use either.
- Audits: **non-existent** - No mention on the product website, neither found any through a quick search on the web.

There is no apparent concern about security on their websites, which is certainly not good. Furthermore, there is apparently an SSL configuration problem with their corporate website, which is not a good sign.

Connectivity/Flexibility

- Number of compatible hardware platforms: **4** - Android, Arduino, Raspberry Pi, and Generic Java-capable board (Cf. dedicated documentation page). Portentially highly compatible with many other platforms, but since no detail is given, we may assume with some reserves that not so many have actually been tested.
- Number of supported protocols: **8** - MQTT, AMQP, OpenWire, XMPP, HTTP REST requests, WebSocket, Hazelcast, Stomp (Cf. dedicated documentation page).
- Number of SDK implementations: **2** - Android and IOS.
- Modularity: **good** - Design built out of preexisting opensource components. DB apparently easily changeable between MongoDB and Apache HBase (Cf. dedicated documentation page)

Overall, many connectors available, though seem to be a little weak on the hardware side, and lack of a SdK for Desktop applications.

Scalability

- Third-Party scalability: **excellent** - Sitewhere offers a choice between MongoDB, Apache HBase and InfluxDB, which are three highly scalable Data Storage Technologies.
- Platform scalability: **non-existent** - It seems a Sitewhere instance has no way to communicate with another and allow load-balancing.

It seems that Sitewhere's scalability claim is only based off the capabilities of its 3rd-parties components.

Arduino/NodeMCU compatibility

- Library: **good** - Quite a lot of work seems to have been put into arduino compatibility, and the library seems very complete (with several example sketches included). However, no particular documentation about the NodeMCU/ESP8266.
- Boilerplate code: **good** - The event/publish/subscribe structure, while being very efficient, requires here quite a lot of boilerplate code just to send a little bit of data.

Quite good Arduino compatibility, but it would really be welcome to have some documentation on its usage with NodeMCU/ESP8266 specifically.

Java

- Server percentage: **81.8%** - The server is mainly just plain Java. Exactly what we need.

3.2 Kaa

Project Health

- First Release Date: **2014/06/30** - The Kaa Project seems to have been open-sourced from its very beginning.
- Latest Release Date: **2016/10/28** - The latest release dates back to the previous year, and could have us think that the project's activity has significantly dropped. The project has an irregular release cycle, "When it's ready"-Debian-like.
- Latest Commit Date: **2017/06/31** - Actually, it seems that the project is still very much active, and a recently posted video on Youtube shows that they plan to release the 1.0 version during this summer. They seem to have reoriented all of their efforts into documenting/testing the project for the upcoming release.
- Number of main contributors: **10** - There is a reasonable number of main contributors, whose work is rather well-distributed along time. All in all, the project has received contributions from 57 different contributors, which shows a good interest from the IoT community.
- Open issues ratio: **0.0173032153** - A very low ratio, almost equal to zero : over the course of the project, 15893 issues have been opened, and now only 275 remain. It certainly goes to show that they have a proactive behaviour toward bugs. It is to be noted that they don't use Github Issues, but their own Jira instance.

Kaa has a very good project health overall, especially since it has reached a point where the team behind it can trustfully say that they are ready for the 1.0 release.

Company backing

- Company Support: **good** - The company behind Kaa, KaaIoT, seems to be more committed to corporate rather than community support (see this StackOverflow discussion), especially with the upcoming release.
- Company Adoption: **average** - Given that the company has existed for 3 years, has, acccording to its LinkedIn page, between 51 and 200 employees, and that they are still recruiting a lot, we can assume they have definitely found clients to buy one of the many formulas they offer on their corporate website. However, no trace was found on who these clients are.

Kaa is supported by a rather big team, and after 3 years of continued existence, we may assume that its company backing is good.

Documentation

- Currentness: **good** - It seems like they have well divided and updated the documentation for each version. Though, it has the same problem as Sitewhere : you have to go to the repository's documentation folder.
- Adaptability: **excellent** - The documentation itself is very-well separated by role of the reader. You have everything: administrator, hardware programmer, contributor guides. Each of these offer detailed explanations, with plenty of screenshots and code snippets to help you get started.

The overall quality of the documentation is very good, and makes you feel you can easily start using the solution right away.

UI-UX

- Server Management UI: **good** - From what can be seen in the documentation, the administration UI is simple but complete, with a not so modern look, but we can bet that will be heavily upgraded in version 1.0.
- Sample applications: **excellent** - Kaa offers a plethora of sample apps that will help you get started, be it with hardware integration or client (apps) integration.

The user experience is especially good thanks to the many sample applications, and the clearly displayed will the Kaa project has to be extremely compatible.

Security

- Statements: **good** - A clear concern for security has been expressed by the CTO of the company in a blog post and is also mentioned in their FAQ and documentation. According to these sources, data communication between the components is secured using with 2048bits-RSA encryption and 256bits-AES signing. Apparently, Kaa also ensures secure data storage by enforcing database-level encryption, as well as tenant-level encryption, which is very good practice. In the FAQ, it is also said that Kaa is fault-tolerant, since the data is automatically replicated across the nodes.
- Audits: **non-existent** - No mention on the product website, neither found any through a quick search on the web.

A clear concern for security is expressed, and some of the measures taken in favor of it are clearly explained. However, no independent security audit of the solution has ever been executed.

Connectivity/Flexibility

Note: With the incoming 1.0 version, connectivity and flexibility is apparently about to skyrocket, since according to the Early Access application page, the platform will turn to a SDK-less, technology independent paradigm, with Out-of-the-box MQTT support.

- Number of compatible hardware platforms: **8** - Kaa provides 8 endpoint SdKs for many popular hardware platforms : RaspberryPi, ESP8266, ...
- Number of supported protocols: **1** - Devices must use Kaa's own protocol (described here), based on MQTT/CoAP. For easy implementation, Kaa provides endpoint SdKs for the many different platforms.
- Number of SDK implementations: **5** - Kaa provides 5 endpoint SdKs for all main OSes : Linux, Android, MacOS, Windows, and finally, a generic SdK written in Java.
- Modularity: **good** - it is already possible to change some components of Kaa pretty easily, for example, MongoDB and Cassandra are both supported as NoSQL DBs, but it is claimed that for the 1.0 version "Every component of the platform can be customized or substituted with 3rd party software".

Kaa overall boasts a very good connectivity and flexibility, and it seems they will pursue in this direction.

Scalability

- Third-Party scalability: **excellent** - Kaa offers a choice between two NoSQL DBS, MongoDB and Cassandra, and two SQL DBS, MariaDB and PostgreSQL, which are highly scalable.
- Platform scalability: **excellent** - Kaa has been thought as a cluster of server nodes, and uses Apache ZooKeeper to coordinate services. Kaa also includes load-balancing capabilities. (Cf. its Architecture Overview)

Kaa seems to have been truly been built from the ground-up to be fully distributed.

Arduino/NodeMCU compatibility

- Library: **good** - As said before, there is a dedicated SdK for the ESP8266, but not for the NodeMCU, neither Arduino.
- Boilerplate code: **good** - There is quite a little boilerplate C code, but it is well-explained, and it shouldn't be really too hard to work with it.

Relatively good compatibility with the ESP8266, however, the support for Arduino or NodeMCU is lacking.

Java

- Server percentage: **69.9%** - Mainly developped in Java (Netty+Spring), almost all of its third party services are written in Java too.

3.3 Device Hive

Project Health

- First Release Date: **2013/09/19** - The first release fo the server software has been made in 2013, though, on the corporate website, the copyright indicates an existence starting in 2012. The project has undergone big changes at its beginning, switching from .NET to Java, according to this article.

- Latest Release Date: **2017/08/04** - The last release is very recent, meaning it still actively supported. The project has an irregular release cycle, "When it's ready"-Debian-like.
- Latest Commit Date: **2017/08/17** - As of the writing of this report, commits are made daily on the development branches: we can thus deduce that the project is still undergoing developments.
- Number of main contributors: **5** - It seems that two of the original contributors of the project do not contribute anymore to the server: maybe they ended up managing it. Anyway, the code base seems rather shared among a few people, which is good. A total of 30 persons have contributed to the project, which means there is a mild interest from the FOSS IoT community.
- Open issues ratio: **0.3434343434** - The project doesn't seem to use Github issues a lot: since its creation on Github, only 99 issues have been created and 34 are still open to this day.

The project seems in overall good health, and still growing.

Company backing

- Company Support: **good** - DeviceHive is supported by DataArt, a company that has, according to LinkedIn, 1001 to 5000 employees. Community support seems rather weak, with only a helpdesk that doesn't seem to have been of much use (only one message), and the sporadic use of Github issues doesn't clue at a very active community exchange.
- Company Adoption: **good** - DataArt has a huge list of clients, and received quite a few awards, but most of this is not directly linked to DeviceHive, which seems to be only one of their many projects. On their success stories pages, there is actually one company that mentions DeviceHive: Wot.io.

Overall, DeviceHive has good company backing, though the community aspect of the project seems lacking.

Documentation

- Currentness: **good** - There is a separate documentation for the older 2.0 and latest 3.3.0 versions, which tends to prove that the documentation is up to date, however, there are no easy means to check the last edition dates since it is not even on Github but on a proprietary service called ReadMe.
- Adaptability: **average** - There is no separation of roles in the documentation, but rather a collection of categories, which goes directly from the "Getting Started and tutorials" to "Specific technical information". There is no page adressing the global architecture.

UI-UX

- Server Management UI: **good** - The server management has a beautiful design, and even comes with an integrated tutorial that easily lets you get started. However, it seems it has rather few capabilities (you can only manage three basic tables: networks, devices and JWT tokens, but there is nothing to configure the server graphically).
- Sample applications: **average** - The only client interface provided is a web interface, based on FreeBoard. Nothing for native development. As for sample apps, there is only one for RaspberryPi and one for ESP8266.

Security

- Statements: **poor** - No statements on security whatsoever. It is not even clearly documented what they use for authentication, encryption and signing. Nothing on the corporate website, nothing in the documentation, except an API for authentification which means there is something, but that's it.
- Audits: **non-existent** - No mention on the product website, neither found any through a quick search on the web.

Security is not clearly a concern for the DeviceHive project, which is quite bothering for an IoT project.

Connectivity/Flexibility

- Number of compatible hardware platforms: **2** - Only two platforms have been documented : RaspberryPi and ESP8266.
- Number of supported protocols: **3** - REST API, WebSockets or MQTT. This is the minimum to potentially allow for any kind of device to connect, from OS-enabled ones, to barebone ones.
- Number of SDK implementations: **2** - Python, Node/Javascript. It seems there were many more before (Go, Java,...), but they have now been deprecated.
- Modularity: **poor** - Though DeviceHive is built on top of popular third-party FOSS components (ElasticSearch, Apache Spark, Cassandra and Kafka), no choice is left as to what to use.

The connectivity and flexibility of DeviceHive seems in fact rather poor overall.

Scalability

- Third-Party scalability: **excellent** - The scalability of the components named above (ElasticSearch, Apache Spark, Cassandra and Kafka) is very good, as they were all designed for this.
- Platform scalability: **good** - DeviceHive uses a container based service oriented architecture approach, managed and orchestrated by Kubernetes. This takes advantage of the inherent capabilities of Docker, but it is not mentioned how the different DeviceHive instances might interact (as their is no Architecture overview in the documentation!).

The approach DeviceHive takes to scalability definitely is interesting, but also cruely lacks documentation.

Arduino/NodeMCU compatibility

- Library: **average** - No Arduino nor NodeMCU libraries, but there is one for the ESP8266.
- Boilerplate code: **good** - Minimal boilerplate code, since it is javascript.

There are some examples with the ESP8266, however they do not seem to cover much.

Java

- Server percentage: **97.2%** - Almost everything is pure Java, which makes it potentially easily maintainable.

3.4 Zetta JS

Project Health

- First Release Date: **2014/07/23** - The project has started on Github three years ago, which is more or less the same as most of the projects presented here.
- Latest Release Date: **2017/07/07** - The latest release is little more than one month old : we can deduce of this that the project is still maintained.
- Latest Commit Date: **2017/07/07** - No advancements have been made since the last release: maybe is it due to summer holidays.
- Number of main contributors: **3** - Only three people are behind this project, and it seems that one of them as left the project at the end of 2016. The Github organization only belongs to one person, which is not necessarily a good thing.
- Open issues ratio: **0.2565789474** - The project doesn't seem to use Github issues a lot: since its creation on Github, only 152 issues have been created and 39 are still open to this day.

Project health looks greenish overall. It might lack maintainers, which would explain the high open issues ratio.

Company backing

- Company Support: **good** - ZettaJS is backed up by APIGEE, a company that creates API-driven applications, and that is been bought by Google in 2016. Actually they sell Apigee Link, a service that leverages Zetta, adding enterprise support and cloud services. As for community support, there is a relatively active dedicated Google Group.
- Company Adoption: **average** - APIGEE seems to have a lot of big customers, and quite a few success stories. However, none of them seem directly related to Zetta.

Overall good company backing, though we have no clear and direct evidence of companies using Zetta.

Documentation

- Currentness: **average** - Most of the wiki hasn't been updated since last year, thus we can infer that there may be a few elements that may differ with the current release.
- Adaptability: **average** - There is no separation of roles in the documentation, but rather a collection of categories, which goes directly from the "Getting Started and tutorials" to "Specific technical information". It definitely feels more like something oriented for tinkerers, rather than to the many different actors using an IoT platform.

Overall, the documentation doesn't look that good. Particularly, it uses the integrated Wiki function of Github, making it clearly dev-oriented, and ignoring all other actors.

UI-UX

- Server Management UI: **average** - There is a UI but it seems only capable to list devices and access their APIs. No user management, no tenant management,...
- Sample applications: **average** - There are a few example projects that are relatively well-documented.

In terms of UI-UX, ZettaJS seems lacking overall, since its focus is primarily in providing APIs.

Security

- Statements: **poor** - No statements about security except one short paragraph in the documentation, that doesn't explain the security strategy at all.
- Audits: **non-existent** - No mention on the product website, neither found any through a quick search on the web.

ZettaJS doesn't express much concerns about security, though it is a key element of IoT.

Connectivity/Flexibility

- Number of compatible hardware platforms: **11** - This is hard number to estimate since Zetta rather use a generic concept of "Driver" for any kind of connection. From the official driver list, the following harware platforms have been spotted: Wemo, Sphero, Spark, BeagleBone, Pinoccio, Pebble, OpenXC, RaspberryPi, XiaoMi, Intel Edison, Google Glass.
- Number of supported protocols: **1** - ZettaJS seems to only understand HTTP requests, though it claims it should be able to bridge other protocols to it, but we couldn't find any such example.
- Number of SDK implementations: **1** - There is only a native SDK for IOS.
- Modularity: **excellent** - As everything is API, any component can be added/changed/removed easily.

The API-driven ideology seems to make connexion to the system very easy, but in fact, to this day, there seem to be no easy way to connect devices that don't run a linux kernel on them.

Scalability

- Third-Party scalability: **poor** - LevelDB doesn't seem to have any sort of distribution capabilities.
- Platform scalability: **poor** - As all the servers can easily communicate and exchange data through HTTP APIs, the project seems rather scalable, but there is no distributed data storage, or integrated load-balancing.

ZettaJS doesn't seem to integrate anything related to scalability.

Arduino/NodeMCU compatibility

- Library: **poor** - No official library provided either for Arduino, NodeMCU or ESP8266, but there is a beginning of an implementation here.
- Boilerplate code: **poor** - No official boilerplate code either, but there is a beginning of an implementation here.

No clear support of Arduino/NodeMCU in ZettaJS.

Java

- Server percentage: **0%** - ZettaJS, is, as the name indicates, a full JS solution. No Java here.

3.5 Parse

Project Health

- First Release Date:
- Latest Release Date: **2017/06/19** -
- Latest Commit Date: **2017/06/20** -
- Number of main contributors: **1** -
- Open issues ratio: **0.0534653465** -

Company backing

- Company Support: **excellent** -
- Company Adoption: **good** -

Documentation

- Currentness: **good** -
- Adaptability: **excellent** -

UI-UX

- Server Management UI: **good** -
- Sample applications: **good** -

Security

- Statements: **average** -
- Audits: **non-existent** -

Connectivity/Flexibility

- Number of compatible hardware platforms: **4** -
- Number of supported protocols: **8** -
- Number of SDK implementations: **2** -
- Modularity: **good** -

Scalability

- Third-Party scalability: **excellent** -
- Platform scalability: **excellent** -

Arduino/NodeMCU compatibility

- Library: **good** -
- Boilerplate code: **good** -

Java

- Server percentage: **81.8%** -

3.6 Blynk

Project Health

- First Release Date:
- Latest Release Date: **2017/06/19** -
- Latest Commit Date: **2017/06/20** -
- Number of main contributors: **1** -
- Open issues ratio: **0.0534653465** -

Company backing

- Company Support: **excellent** -
- Company Adoption: **good** -

Documentation

- Currentness: **good** -
- Adaptability: **excellent** -

UI-UX

- Server Management UI: **good** -
- Sample applications: **good** -

Security

- Statements: **average** -
- Audits: **non-existent** -

Connectivity/Flexibility

- Number of compatible hardware platforms: **4** -
- Number of supported protocols: **8** -
- Number of SDK implementations: **2** -
- Modularity: **good** -

Scalability

- Third-Party scalability: **excellent** -
- Platform scalability: **excellent** -

Arduino/NodeMCU compatibility

- Library: **good** -
- Boilerplate code: **good** -

Java

- Server percentage: **81.8%** -

3.7 Kapua

Project Health

- First Release Date:
- Latest Release Date: **2017/06/19** -
- Latest Commit Date: **2017/06/20** -
- Number of main contributors: **1** -
- Open issues ratio: **0.0534653465** -

Company backing

- Company Support: **excellent** -
- Company Adoption: **good** -

Documentation

- Currentness: **good** -
- Adaptability: **excellent** -

UI-UX

- Server Management UI: **good** -
- Sample applications: **good** -

Security

- Statements: **average** -
- Audits: **non-existent** -

Connectivity/Flexibility

- Number of compatible hardware platforms: **4** -
- Number of supported protocols: **8** -
- Number of SDK implementations: **2** -
- Modularity: **good** -

Scalability

- Third-Party scalability: **excellent** -
- Platform scalability: **excellent** -

Arduino/NodeMCU compatibility

- Library: **good** -
- Boilerplate code: **good** -

Java

- Server percentage: **81.8%** -

4 Summarized analysis

5 Conclusion

- Annexes -