Home Assistant

Kathy Do, Nicholas Francis, Alvi Mahadi, Aigerim Mashkanova

1.0 Introduction

1.1 Purpose

This is Team 11's final project in partial fulfillment of Spring 2019 SENG480A class at the University of Victoria. The purpose of which was to analyze the software architecture of an open source project on GitHub: outlining their stakeholders and business goals, determining their architecturally significant requirments and relevent quality attribute scenarios, developing a module and component and connector view for their architecture based on paramount QAS, conducting a software quality and technical deby analysis. All of which is shown in this document.

1.2 About Home Assistant

Home Assistant, when deployed on your home network using a home server or raspberry pi, can be used to monitor, control and automate (ie. create routines for) all the electronic “smart” devices in your home. These range from climate control and home security devices to coffee makers and light switches. Home Assistant integrates the control of all these devices into one clean User Interface.

1.3 Our Team

Our team consists of four University of Victoria students. We all share a passion for software and more specifically, things written in python. Our Grad Students: Alvi Mahadi and Aigerim Mashkanova. Our Bachelor Students: Kathy Do and Nicholas Francis.

2.0 Business Orientation

To get a sense of the audience for the architectual documentation we were to create for Home Assistant, we did an analysis on their stakeholders and business goals. The results of this analysis are presented in this section.

2.1 Stakeholders

Table 1. Stakeholders of Home Assistant

Role	Concerns	Instances
Acquirers	Oversee the procurement of the system or product	Paulus Schoutsen and Ubiquity Networks
Assessors	Oversee the system's conformance to standards and legal regulation	**Lead Developer Team Paulus Schoutsen, Fabian Affolter, Pascal Vizeli, Otto Winter, Andrew Castille with a total of 32 home-assistant members, community and open-source developers (via GitHub):** As an open-source project, all developers are responsible for conformance checking, to assure that every implementation adheres to and is consistent with the Home Assistant architecture.
Communicators	Explain the system to other stakeholders via its documentation and training materials	**Home Assistant community, open-source developers:** The documentation, demo and tutorials were updated on Home Assistant's official website, which covers beginner to advanced topics around the installation, setup, configuration and use of Home Assistant. On GitHub, 'home-assistant.io' repository contains all Home Assistant documentation mainly contributed by Fabian Affolter, Paulus Schoutsen and Brandon Mathis. Also, if users need any help, they can get in touch with the Home Assistant community in various ways: forum, Discord Chat Server, Twitter, Facebook community, Google+ community and Reddit.
Developers	Construct and deploy the system from specifications (or lead the teams that do this)	**See Assessors:** This open-source home automation platform were founded by Paulus Schoutsen and developed by over 1300 open-source contributors on GitHub. Besides, a team of three: Paulus Schoutsen, Pascal Vizeli and Ben Bangert have contributed to support Home Assistant with a cloud service. Additionally, developer team has developed Hass.io, which is an all-in one-solution and has a user interface used from Home Assistant frontend.
Maintainers	Manage the evolution of the system once it is operational	**See developers.** Developers of the system, mainly the core Home Assistant development team are responsible for the maintenance of the system.

Role	Concerns	Instances
Production Engineers	Design, deploy, and manage the hardware and software environments in which the system will be built, tested, and run	**Server deployment script:** `config.py` file which is called in the server script is developed by Andrey, Johann Kellerman and 21 more developers. **Environment setup:** `setup.py` file which is called to install and manage all dependencies is written by Ville Skyttä, Matthias Urlichs and 8 other developer. **Virtualization:** `Docker image` of this repository is scripted and maintained by Michaël Arnauts, Alex and 20 other developers. `Vagrant` of this project is developed by Tabakhase and Fredrik Baberg. Everything is supervised by Paulus Schoutsen
Suppliers	Build and/or supply the hardware, software, or infrastructure on which the system will run	Website is powered by: Jekyll, Oscalite themes, Hosted in Netlify. Source code is hosted by Github.
Support Staff	Provide support to users for the product or system when it is running	**See developers.** Home Assistant core development team. The Home Assistant development provides support through the Home Assistant forum, Discord chat server, Twitter, Facebook, Google+, and Reddit.
System Administrators	Run the system once it has been deployed	**See users.** Each user manages their own system after deployment.
Testers	Test the system to ensure that it is suitable for use	**See developers.** Developers of the system are required to conduct unit tests on code as written in their docs.
Users	Define the system's functionality and ultimately make use of it	Home owners who wish to implement smart technology into their homes. Hobbyists interested in smart technology. Commercial businesses that want to automate specific business processes.

2.2 Business Goals

Business goals involve the business prospects of a project. The following topics can well be the business prospect of Home Assistant.

Privacy

One of the main goals of home assistant is to make the user data private and secure. The lead developer of this project started this project with the aim of keeping the user data private. All the user data will be stored locally.

Giving the control to the user

All logic will run locally. The user data are all stored locally. The cloud will only be invoked as needed. So the users feel that they have more control on the functionality.

Open source

The platform is freely available for users and companies alike. The more people that choose a privacy focused platform over a cloud-based one is considered to be a win for them.

Interoperability

The platform implements APIs to easily share the data. The system wants the user data to be available to any other application that the user wishes. The project can be extended further to make use of the user data. That provides a great opportunity to expand the business.

Partnerships

Home Assistant has partnerships with several Hardware Manufacturers, Software Development Companies and Cloud Service Providers. Some of them are:

1. Nebu Casa is a subscription based cloud solution of Home Assistant providing deployment, control, maintenance and automation of the Home Assistant project. Partnership with Nebu Casa provides Home Assistant with sufficient funding to continue with the open source work.
2. There are plans to offer a paid cloud Home Assistant subscription, which lets it interact with Amazon Alexa and Google Assistant, something not possible with the standard version. This feature is currently in open beta, and uses Amazon's cloud servers.
3. The system is developed using a modular approach so that support for other devices and operations can be implemented easily. The Home-Assistant author and lead developers always invite and welcome other developers to contribute to this project. This is a strategy to have more devices covered and more functionality implemented to have more users. This fulfills one of their primary goals of having a vast number of users to expand their business.
4. The developers are trying to make it more accessible to a vast number of users by various platform integrations such as Mozilla's Web Things API to make it easier to interact with other home automation platforms. The introduced device management makes it perfectly aligned with the Web Things data model.

3.0 Architecturally Significant Requirement (ASR) Analysis

Utilizing the results from our analysis of their business orientation, we then isolated the key quality attributes that are yielded from the architecture of Home Assistant. Using the attributes, we then came up with a list of ten architectually significant requirments.

3.1 Listing of ASRs

By the definition, "An architecturally significant requirement (ASR) is a requirement that will have a profound effect on the architecture that is, the architecture might well be dramatically different in the absence of such a requirement." We've attempted to ascertain these requirements by going through Home Assistants's documentation and online blogs, as well as developer discussions in forums and various channels. Using the infomation we gathered, we identified several Quality Attribute Scenarios (QAS) for our project. An analysis of our QASs resulted in the following ASRs.

1. The system must have two factor authentication to prevent malicious access.
2. The system requires an SSL certificate when the user wants to operate from online to secure the connection.
3. The system must follow a modular approach to encourage more developers to integrate more modules without breaking the rest of the system.
4. The system registers and processes user commands correctly at a rate of 99.99%.
5. The system provides visual feedback in the event of a user error ex. Error Messages.
6. The system must ensure that the user data is not modified by any third parties by using a digital signature technique.
7. The system must log the failures and reboot itself in less than 5 seconds.
8. The system has the ability to recover and store data when the system crashes.
9. The system allows a user to learn how to use a feature using their intuition in under 2 minutes
10. The system logs system events for monitoring.

3.2 Quality Attribute Scenarios (QAS)

This section first outlines three QAS that our team felt most significant to Home Assistants architecture, then goes on to describe five additional QAS that are largely significant, but runners up to the three templated.

Authentication

Table. 2: MFA QAS

Aspect	**Details**
Scenario Name	Multifactor Authentication
Business Goals	Protect the system and secure user data
Quality Attributes	Authentication
Stimulus	Access restriction
Stimulus Source	Malicious entities (Users, Software)
Response	Allow or block acccess
Response Measure	Successful 99.99%

Reliability

Table. 3: Event Logging QAS

Aspect	Details
Scenario Name	Event Logging
Business Goals	Identify what caused a failure, keep track of user history
Quality Attributes	Reliability
Stimulus	Need for system monitoring
Stimulus Source	Active internal devices Ex. Sensors
Response	Write operation to log file
Response Measure	Every event is logged

Modifiability

Table. 4: Modularity QAS

Aspect	Details
Scenario Name	Integrate additional module
Business Goals	High device coverage and encourage open source contribution
Quality Attributes	Modifiability
Stimulus	Wish for the integration of an additional device, platform, new functionality
Stimulus Source	Developer
Response	Modification is made with no side effect
Response Measure	Zero other modules affected

Data Integrity

The security of user data is one of the most important business as well as technical goals of home-assistant. The author of the repository does not want the big companies to have all the user data just the way they are doing it right now. They provide cloud solutions to problems and in exchange, collect user data and use them to improve the online experience of that user. In the case of home-assistant, all the logic is run and user data is stored locally. Moreover, all the activities are maintained with modern data integrity protocols. This is a very significant architectural requirement of this project.

User Interface

The system must have a very simple yet intuitive interface. The interface should be very responsive with the quality of getting the task done in fewer actions from the user.

Error Feedback

The system must give appropriate error feedback to the user who is trying to integrate the system in a home environment or someone who is trying to code a new module to the system.

Response to System Failure

The system should be available all the time to the user as it is going to be deployed to their home environment. The system must be very responsive in the event of system failure.

Exchange of information

The system must have an easy way to exchange information with other systems to ensure interoperability. Interoperability is one of the most important business goals of the system and this easy data exchange mechanism should be built into the system.

3.2 Utility Tree

Shown below in Figure. 1 is the Utility Tree we made to model our QAS's. Each QAS is ranked on technical difficulty and business priority as per the legend of the figure.

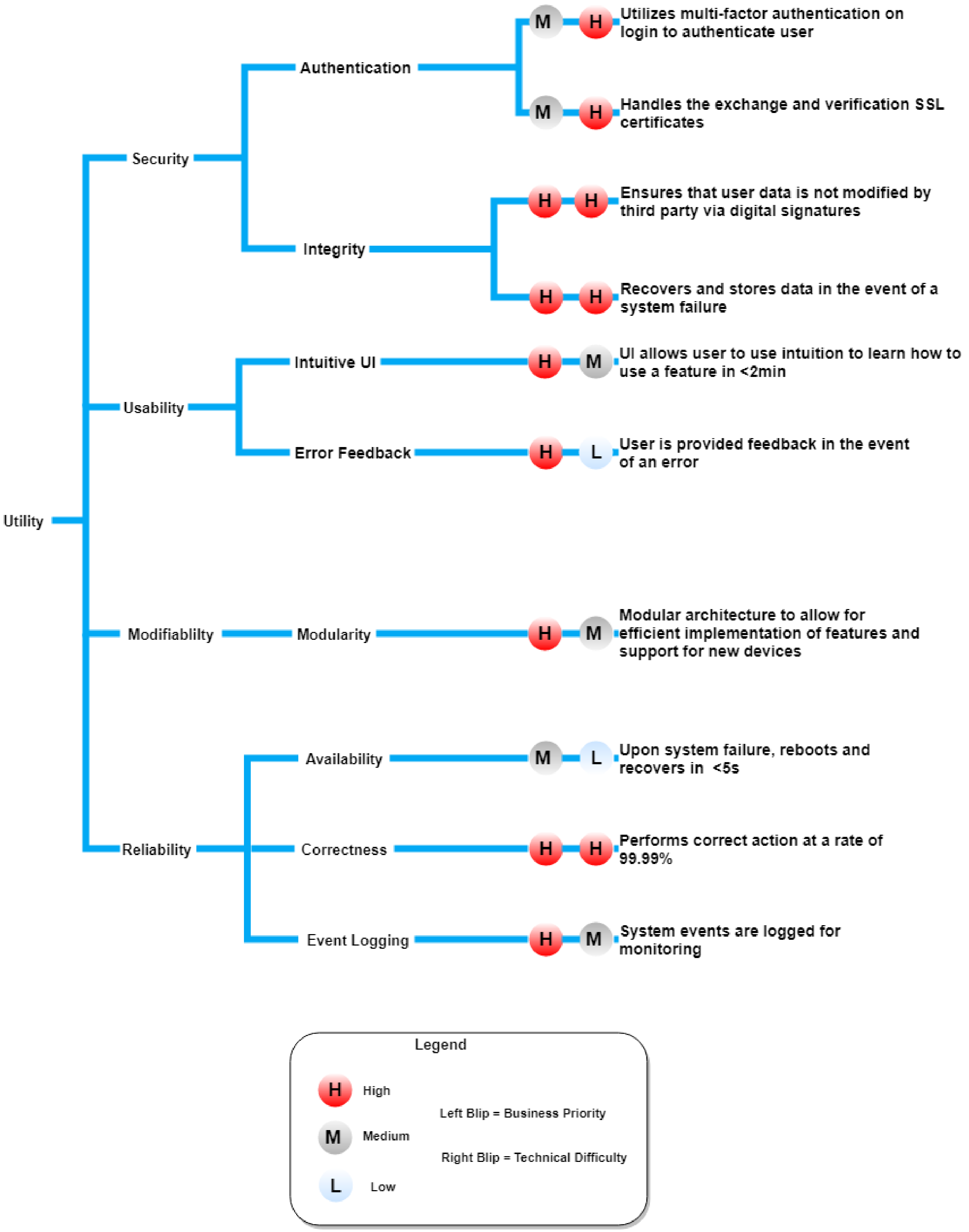

Utility
Security
Authentication
M H Utilizes multi-factor authentication on login to authenticate user
M H Handles the exchange and verification SSL certificates
Integrity
H H Ensures that user data is not modified by third party via digital signatures
H H Recovers and stores data in the event of a system failure
Usability
Intuitive UI
H M UI allows user to use intuition to learn how to use a feature in <2min
Error Feedback
H L User is provided feedback in the event of an error
Modifiablilty
Modularity
H M Modular architecture to allow for efficient implementation of features and support for new devices
Reliability
Availability
M L Upon system failure, reboots and recovers in <5s
Correctness
H H Performs correct action at a rate of 99.99%
Event Logging
H M System events are logged for monitoring
Legend
H High
M Medium
L Low
Left Blip = Business Priority
Right Blip = Technical Difficulty

Figure. 1: Utility Tree for QAS Analysis

4.0 Module View

4.1 Primary Presentation

Immediately below this prelude resides our primary presentation for the module view of Home Assistant's auth module. This view uses combination of view styles allows our primary presentation to show the integral elements of the auth module while indicating the inheritance and dependency relationships between them. It's essentially a class diagram that depicts "isa" relationships and internal dependencies.

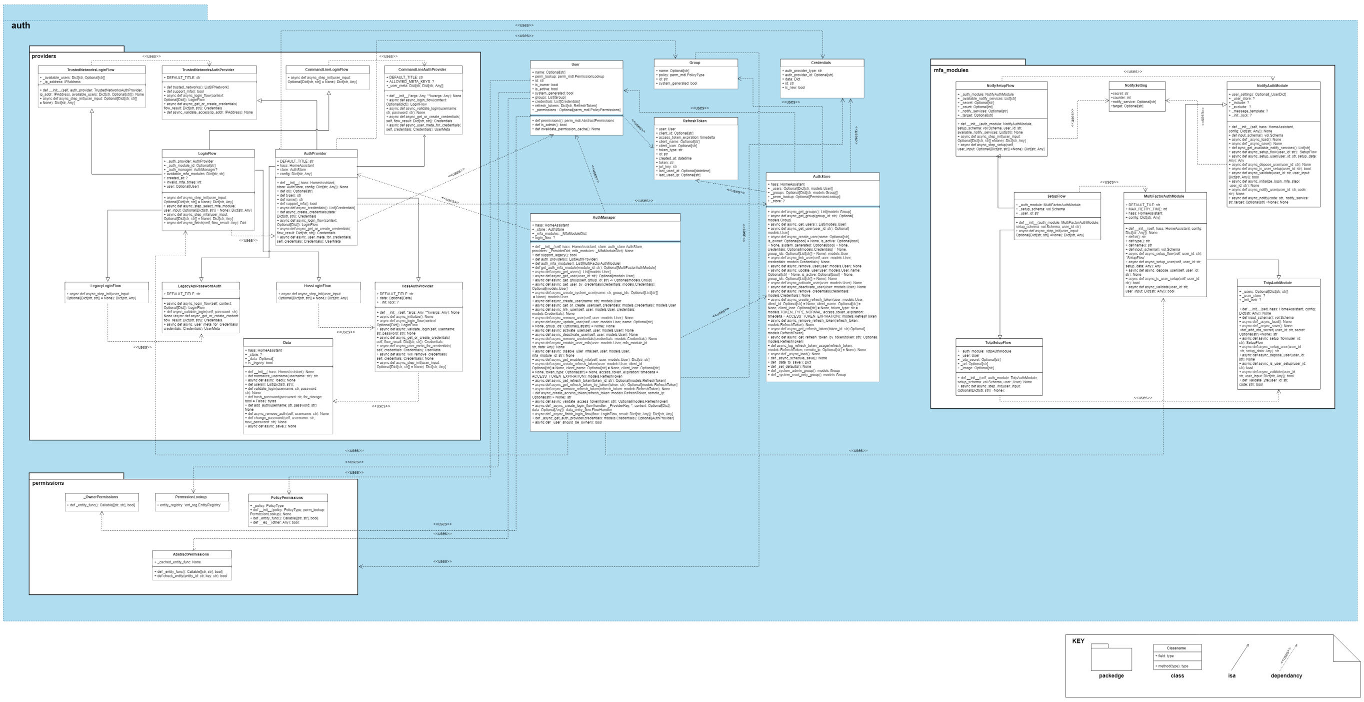


Figure. 2: Primary Presentation of Module View for Auth Module

4.2 Element Catalog

1. Authentication Storage: Storage for auth models.
    - AuthStore: Stores authentication info. Any mutation to an object should happen inside the auth store.
    - System Admin Group: Creates system admin group.
    - System Read-Only Group: Creates read only group.
2. Constant: Contains Constants for the auth module.
3. Initializer: Provide an authentication layer for Home Assistant.
    - AuthManager: Manage the authentication for Home Assistant.
4. MFA Modules: Provide Multifactor Authentication.
    - Initializer: Plugable auth modules for Home Assistant.
        - MultiFactorAuthModule: Multi-factor Auth Module of validation function.
    - Notify: HMAC-based One-time Password auth module. Sends HOTP through notify service.
    - TOTP: Time-based One Time Password auth module.
        - Generate QR code: Generate a base64 PNG string represent QR Code image of data.
        - Generate secret and QR code: Generate a secret, url, and QR code.
        - TotpAuthModule: Auth module validate time-based one time password.
        - TotpSetupFlow: Handler for the setup flow.
5. Models: Authentication model.
    - Group: Defines a group of user.
    - User: Defines a single User.
    - RefreshToken: Refreshes Token for a user to grant new access tokens.

   - Credentials: Credentials for a user on an auth provider.
6. permissions: Manages Permissions and Authorizations.
   - Constants: Contains permission constants.
   - entities.py: Manages permissions policy for an entity.
     - Compile Entities: Compile policy into a function that tests policy.
     - Entity Allowed: Test if an entity is allowed based on the keys.
     - Apply Policy Functions: Apply several policy functions.
   - Initializer: Initializes Permission Policy for Home Assistant.
   - Merge: The merging of policies.
     - Merge Policies: Takes a list of policies, checks compatibility and perform merge.
   - Models: Models for the permissions policies.
     - PermissionLookup: Class to hold data for permission lookups.
   - System Policy: Core system policy including Admin policy and Read-Only policy.
   - Types: Manages the types of the permissions.
7. providers: Manages the Authentication providers.
   - Command line: Auth provider that validates credentials via an external command.
     - InvalidAuthError: Raised when authentication with given credentials fails.
     - CommandLineAuthProvider: Auth provider validating credentials by calling a command.
     - CommandLineLoginFlow: Handler for the login flow.
   - homeassistant.py
     - Data: Holds the user data.
     - HassAuthProvider: Auth provider based on a local storage of users in HASS config dir.
     - HassLoginFlow: Handler for the login flow.
   - Initializer: Initializes authentication providers for Home Assistant.
   - Lagecy API password: Support Legacy API password auth provider.
     - LegacyApiPasswordAuthProvider: Example auth provider based on hardcoded usernames and passwords.
     - LegacyLoginFlow: Handler for the login flow.
   - Trusted Networks: Trusted Networks auth provider. It shows list of users if access from trusted network. Abort login flow if not access from trusted network.
     - InvalidAuthError: Raised when try to access from untrusted networks.
     - InvalidUserError: Raised when try to login as invalid user.
     - TrustedNetworksAuthProvider: Trusted Networks auth provider. Allow passwordless access from trusted network.
     - TrustedNetworksLoginFlow: Handler for the login flow.
8. Utility: Provides utility for authentication.
   - Generate Secret: Generate a secret from secrets.token_hex from python 3.6.

4.3 Context Diagram

From the definition of the textbook, a context diagram "shows how the system or portion of the system relates to its environment". We have shown the primary presentation of the Authentication System of Home Assistant which is a part of the Security QAS. In the context diagram below we have shown the scope and interaction of the Authentication System with respect to the other internal and external entities of Home Assistant.

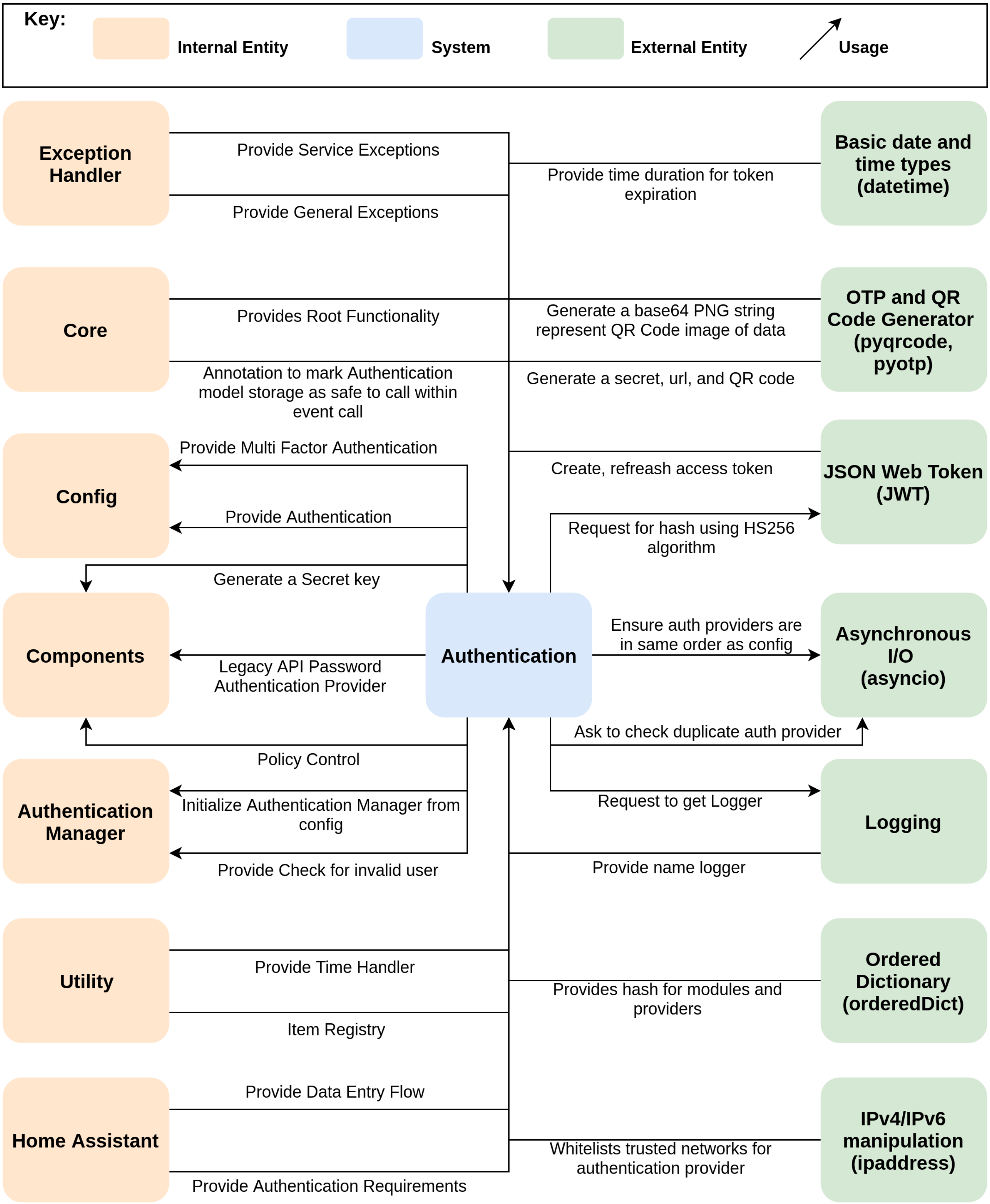

Key:
Internal Entity
System
External Entity
Usage
Exception Handler
Core
Config
Components
Authentication Manager
Utility
Home Assistant
Authentication
Basic date and time types (datetime)
OTP and QR Code Generator (pyqrcode, pyotp)
JSON Web Token (JWT)
Asynchronous I/O (asyncio)
Logging
Ordered Dictionary (orderedDict)
IPv4/IPv6 manipulation (ipaddress)
Provide Service Exceptions
Provide General Exceptions
Provides Root Functionality
Annotation to mark Authentication model storage as safe to call within event call
Provide Multi Factor Authentication
Provide Authentication
Generate a Secret key
Legacy API Password Authentication Provider
Policy Control
Initialize Authentication Manager from config
Provide Check for invalid user
Provide Time Handler
Item Registry
Provide Data Entry Flow
Provide Authentication Requirements
Provide time duration for token expiration
Generate a base64 PNG string represent QR Code image of data
Generate a secret, url, and QR code
Create, refreash access token
Request for hash using HS256 algorithm
Ensure auth providers are in same order as config
Ask to check duplicate auth provider
Request to get Logger
Provide name logger
Provides hash for modules and providers
Whitelists trusted networks for authentication provider

Figure. 3: Context Diagram of Module View for Auth Module

4.4 Behaviour Diagram

The following behaviour diagram is a trace-based diagram that demonstrates the pattern of interaction between objects, and the sequence of actions involved in authenticating a user's login to the system. The sequence diagram below takes a single trip through the authorization process, starting from the user logging into the client website, and the subsequent actions between the client and Home Assistant. After the client authorizes the URL, the user is redirected to `redirect_url` where the Home Assistant authorization provider then returns an authorization code to the client. The client then requests and receives JSON tokens from the Home Assistant authorization provider through HTTP POST requests.

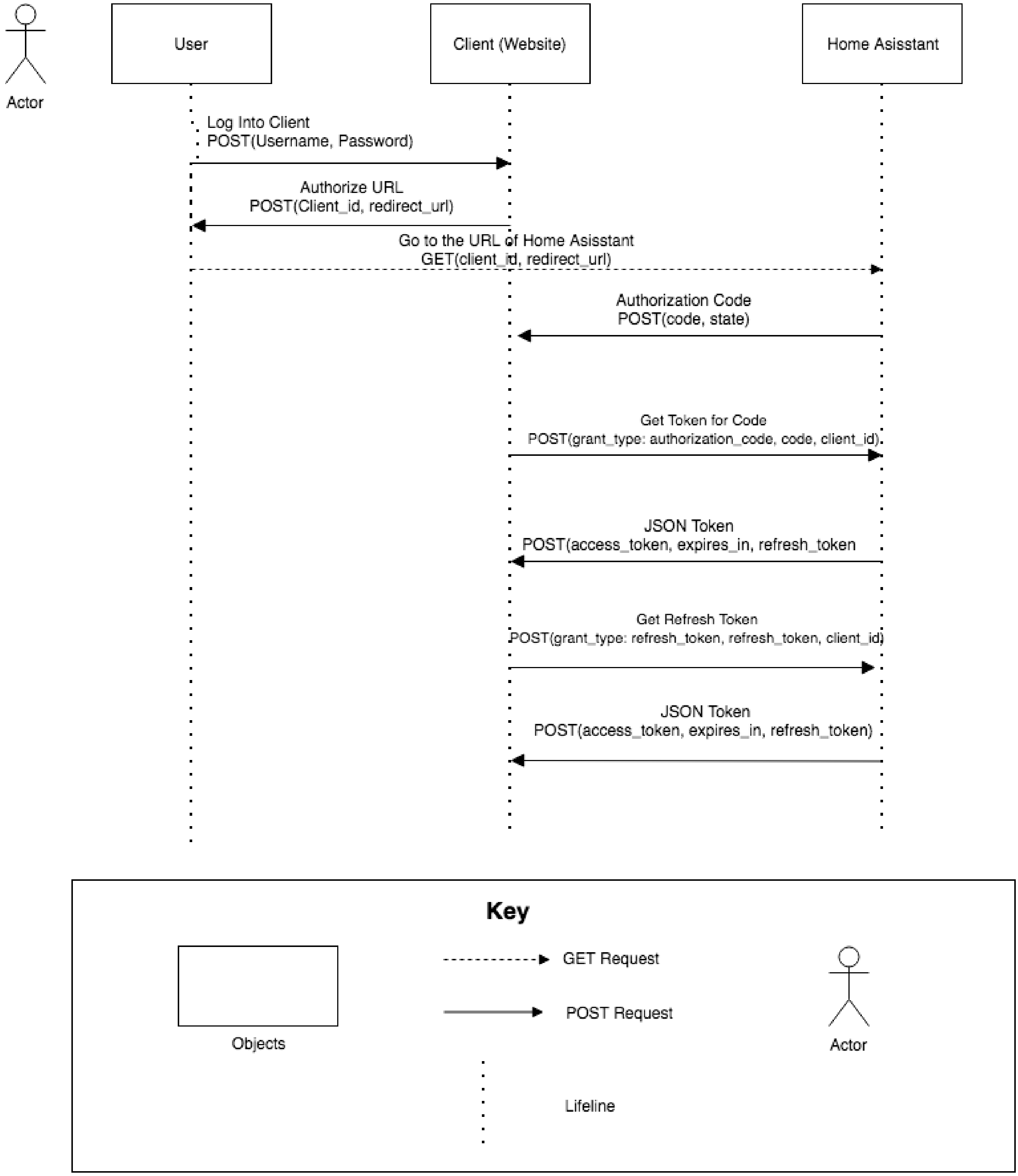

Actor
User
Client (Website)
Home Asisstant
Log Into Client
POST(Username, Password)
Authorize URL
POST(Client_id, redirect_url)
Go to the URL of Home Asisstant
GET(client_id, redirect_url)
Authorization Code
POST(code, state)
Get Token for Code
POST(grant_type: authorization_code, code, client_id).
JSON Token
POST(access_token, expires_in, refresh_token
Get Refresh Token
POST(grant_type: refresh_token, refresh_token, client_id)
JSON Token
POST(access_token, expires_in, refresh_token)
Key
Objects
GET Request
POST Request
Lifeline
Actor

Figure. 4: Behavior Diagram of Module View for Auth Module

4.5 Rationale

The Home Assistant security authentication module, `auth`, is organized under three sub modules: `mfa_modules`, `permissions`, and `providers`. The QAS we identified that is closely associated to this primary presentation deals with authentication. The QAS tests whether the system has authentication implemented with the response measure being the system allowing or blocking access to the user. In the auth main module, we can see that there is standard authentication implemented in the `permissions` and `providers` modules. The `permissions` module describes the user privileges and system policies for the user on a task level. The `providers` module describes the third party authentication providers and trusted network configurations. The `mfa_modules` module contains the implementation of pluggable multifactor authentication modules. A possible reason for why the designers organized the authentication system to utilize a third party authentication provider is to add a layer of security for the user's sensitive information. Since the user is usually hosting their own home assistant system locally, there would be greater risk for the user if they were storing their own credentials. Handing off the authentication to a third party provider takes some responsibility off of the client in case of an external attempt to compromise sensitive data. The standard authentication implemented by the `permissions` and `providers` modules is further extended by the mfa_modules module that adds an extra layer of authentication that could allow or block access to a user.

Another one of our QAS that we described previously was the ability to integrate an additional module which pertains to the modifiability of the architecture. The authentication implementation follows an object oriented approach which provides a modular structure for the system. There is a benefit from using object oriented programming concepts of how objects can be reused within and across applications. This reuse of objects makes the software easier to maintain and enables faster development. More effort has been put into object oriented design and analysis, which lowers the cost of development. We can see the benefits of this modularity if we take the example of adding a new multifactor authentication module. Currently, in home assistant's `mfa_modules` implementation, there are two different multifactor authentication schemes: Time Based One-time Password (TOTP) and HMAC-based One-time Password. The organization of the system modules by `auth` and then `mfa_modules` makes the multifactor authentication modules easy to identify and analyze. A reason for having the modules organized in this way keeps each functionality isolated from each other, so that one can navigate easily to the code that relates to the the new module to see how the additional module can fit. To add and integrate an additional module could then be added and there should not be severe side effects to the rest of the system. In our primary presentation diagram, we can see the dependencies of the authentication modules have no long chains of dependencies, neither are there any circular dependencies. As a result of this, a developer can quickly analyze the code structure to integrate an additional multifactor authentication module under `mfa_modules` quite easily.

4.6 Interface

Table. 5: Interface #1 for Module View

Interface Identity: Authentication Service

Resources provided:
Pre-Conditions:
The client constructs the request URI by adding the following parameters to the query component of the authorization endpoint URI
client_id
`REQUIRED. client identifier`
redirect_uri
`REQUIRED.The URL to which the service will redirect the user after issuing the authorization code. The url: https://www.my-application.io/hass/auth_callback`
state

OPTIONAL. The authorization server includes state value when redirecting the user-agent back to the client

Post-Conditions:

access_token

REQUIRED. The access token issued by the authorization server used to authorize requests and store additional information about the user

token_type

REQUIRED. Type of token, it is also inserted into the request to API

expires_in

RECOMMENDED. This attribute holds the number of seconds before the access token expires

refresh_token

REQUIRED. This attribute issued by the server based on the results of successful authentication

Data types

{Integer} client_id: application client id

{String} redirect_url: the place where the service redirects the browser after the authorization code is provided

{String} state: stores instance url that you the user authenticating with

Error Handling:

- raise RuntimeError('Credential with unknown provider encountered')
  The error occurs when system could not find authentication providers
- raise ValueError('Unable to deactive the owner')
  The error occurs when system tries to deactivate the user
- raise ValueError('System generated users cannot have refresh tokens connected ','to a client.') The error occurs when system tries to create a new refresh token for a user
- raise ValueError('System generated users can only have system type ','refresh tokens')
  The error occurs when system could not find client_id of the user
- raise ValueError('Client_name is required for long-lived access ','token')
  The error occurs cause each client_name can only have one long_lived_access_token_type of refresh token
- HTTP status 400- Bad Request:
  The error occurs when the server can understand the request, but does not allow it to be executed, because the client has access restrictions to the current section.

Quality Attribute:

Security: The interface ensures the quality attribute and use authentication API to authorize users to access to Home Assistant instances .Users also could set a time limit for authorization and limit the number of attempts to enter a password.

Rationale:

While using authentication API with Oauth 2.0 you do not need to worry about ensuring the confidentiality of the user's login and password. Login and password are not transferred to the application, and therefore, they can not fall into the hands of intruders.

Usage Guide:
Home assistant gives the user the ability to easily configure authentication through the authentication API. The API typically uses tokens to authenticate users and does not save session state between requests. The authentication API uses thr OAuth2 server implementation for home assistant application. When a user clicks on a link, they must first log in the service, in order to verify their identity (unless the users have already logged in). Then the service will ask them to confirm whether the application should be allowed or denied access to their account. If the user clicks on "Authorize the application", the service redirects the browser to the application redirection URI that was specified when registering the client, along with the authorization code. If the authorization is successful, the API will send a response to the application containing the access token and refresh token. The access token will have a limited lifetime while refresh tokens will remain valid until a user deletes it. Now the application is authorized!

Table. 6: Interface #2 for Module View

Interface Identity: Multifactor Authentication with Authenticator App

Resources provided:
Pre-Conditions:

- User should registered into the system by giving username and password
- Multi-factor authentication module should be enabled
- MFA module should extend SetupFlow
- The auth module need to provide an implementation of MultiFactorAuthModule class

Post-Conditions:

- secret code generated by home assistant
- validation of session

Data types:
{Integer} user_id: application user id
{String} ota_secret: generated secret code
{String} qr_code: generated qr code
{String} qr_code: generated mfa code from any authenticator app

Error Handling:

- raise ValueError('System generated users cannot enable ','multi-factor auth module.')
The error occurs when the system tries to enable a multi-factor auth module for a user
- raise ValueError('System generated users cannot disable 'multi-factor auth module.') The error occurs when the system tries to disable a multi-factor auth module for a user
- raise ValueError('System generated users cannot have refresh tokens connected ','to a client.') The error occurs when the system tries to create a new refresh token for a user

Quality Attribute:

- Security: The interface ensures the quality attribute and use two-factor authentication to access the system. Multi factor authentication provides secure access to applications and data.
- Modifiability: The interface supports this quality attribute. Based on the user preference, changes in any authentication modules such as `mfa` or `notify`, will not affect the system due to the modules

implemented in different .py files.

Rationale:
The multifactor authentication interface is designed in a way that the user can enter the mfa code from the home assistant dashboard. The multifactor authentication supports different types of Authenticator Apps.

Usage Guide:
Multi factor authentication module gives users the ability to secure their accounts, by linking their login and password for home assistant to any Authenticator App. After enabling mfa module, the qr code is generated automatically by home assistant. Then with the help of Google Authenticator, from qr code, the 6-digit number is generated and must be inserted as an mfa code. Afterwards, the mfa module will be enabled for the user.

5.0 Component and Connector (C&C) View

5.1 Primary Presentation

Component and Connector view is one fundamental view of the software architecture where component is a unit of behaviour that defines what the component can do and what it requires to do that job. Connector on the other hand is an indication that there is a mechanism that relates one component to another usually shown through relationships such as data flow or control flow. We are taking **Event logging** capability of Home Assistant that defines the reliability of the system. We have choosen Pipe and Filter to explain the **System Log** of Home Assistant that achieves this functionality.

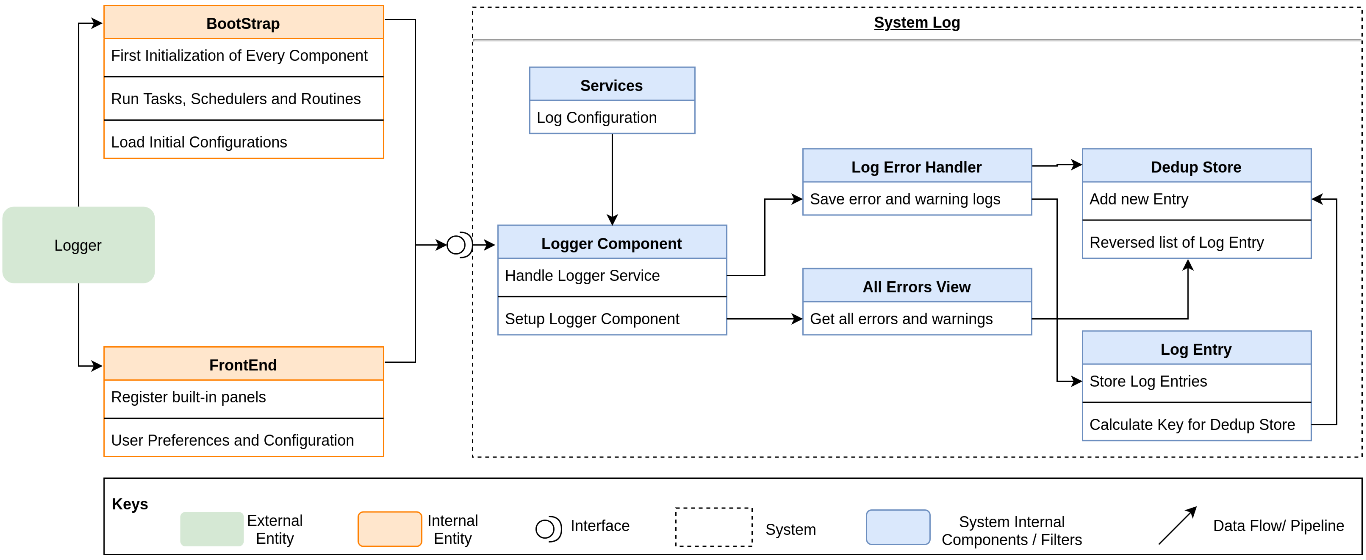


Figure. 5: Primary Presentation of Component & Connector View for System Log

5.2 Element Catalog

1. System Log: The `system_log` component stores information about all logged errors and warnings in Home Assistant.
    - **Services:** The configuration for the system log. This configuration provides the clear and read mechanisms. Also it provides the log structure such as the description, message, level(debug, info, warning, error critical) and the logger name under which to log the message. This is set default to `system_log.external`.
    - **Logger Component:** The interaface to set up the logger components. This component performs tasks like handling the logger services and removing logging handler when Home Assistant is shut

down.

  - **Log Error Handler:** This is the log handler for error messages. This component initializes a new `LogErrorHandler` to save error and warning logs. Everything logged with error or warning is saved in local buffer. A default upper limit is set to 50 (older entries are discarded) but can be changed if needed.
  - **All Errors View:** This component is used to get all logged errors and warnings.
  - **Dedup Store:** This is the data store to hold maximum amount of deduped entries. A new Dedupstore is initialized to add a new entry. This is done by updating the stored entry by removing the first record which should also be the oldest.
  - **Log Entry:** This component store Home Assistant log entries. This component provides initialization for a log entry, calculates a key for Dedup Store and converts object into a dictionary to maintain backward compatibility.

2. BootStrap: This component is used to provide methods to bootstrap a Home Assistant instance. This component gets all the logging information from the `logger` library and the logging system is invoked everytime when there is some exception regarding trying to configure Home Assistant from a configuration dictionary to initialize, install or start any component of the system.
3. FrontEnd: This component handles the frontend for Home Assistant. Every component and event of the system is linked to this component such as Panel, Registration etc. Every event is logged as `info` and `warnings` and exceptions are logged as `error` or `critical`. The debug mode is logged as `debug`.

5.3 Context Diagram

The context diagram below outlines the System Log component and it's scope of interaction with internal and external entities of Home Assistant.

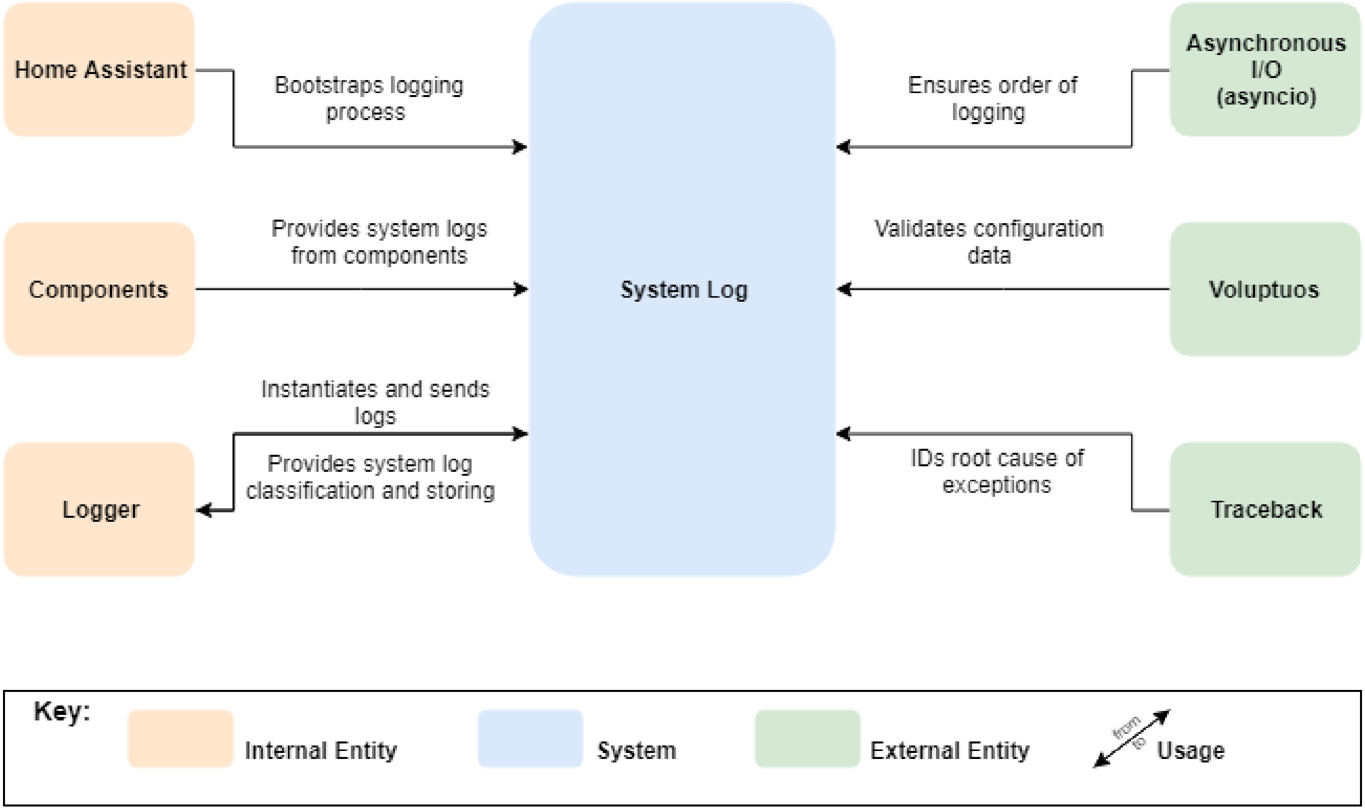

Home Assistant
Bootstraps logging process
Components
Provides system logs from components
Logger
Instantiates and sends logs
Provides system log classification and storing
System Log
Asynchronous I/O (asyncio)
Ensures order of logging
Voluptuos
Validates configuration data
Traceback
IDs root cause of exceptions
Key:
Internal Entity
System
External Entity
from
to
Usage

Figure. 6: Context Diagram of Component & Connector View for System Log

5.4 Behaviour Diagram

The UML state diagram below models the states of the Home Assistant system and its event logging behaviour. In a UML state diagram, each entity or each of sub-entities is always in exactly one of a number of possible states. There are well defined conditional transitions between the states. Many software systems are event-driven, which means that they continuously wait for the occurrence of some external or internal even. In our case, the arrival of an event data packet is the event that triggers the system_log module. After recognizing the event, such systems react by performing the appropriate computation that may include manipulating the hardware or generating “soft” events that trigger other internal software components. We model the `system_log` “soft” events within a composite state that expands upon the log processing state into substates.

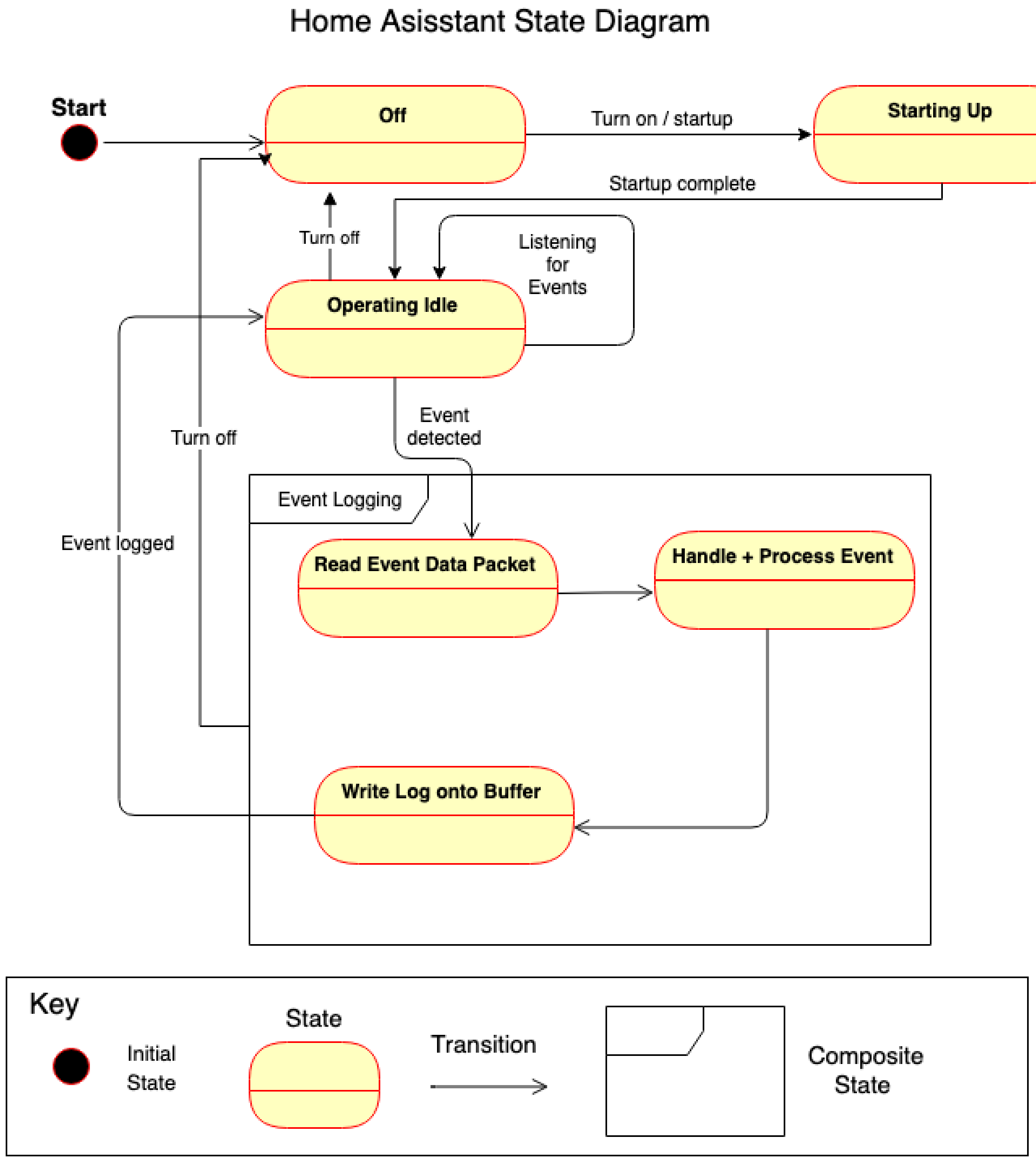


Figure. 7: Behavior of Component & Connector View for System Log

5.5 Variability Guide

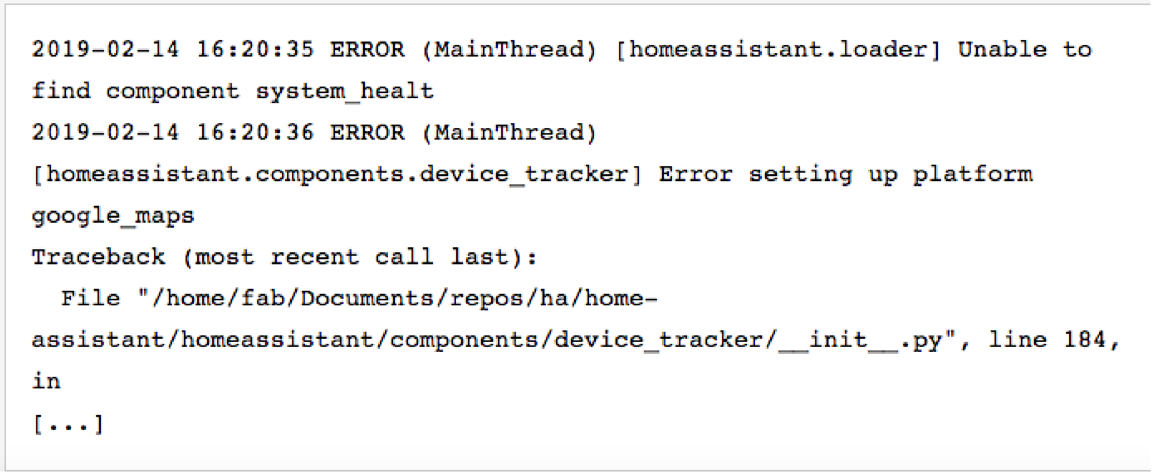
In the above C&C view of the Home Assistant event logging system, the architecture in the view shown will not change dramatically due to optional components, selection of different implementations of an element, nor replication of components and connectors. Being that the event logging system is self contained and contains all of the functionality needed for this task, there is no functional need for additional variability mechanisms. One optional component that was identified was NetDisco, a web based network management tool which discovers local devices devices and services. The method `figure_out_source` in the `system_log` init file checks to see if NetDisco was installed and if so, imports the NetDisco path to use. The main variability in the system that can be changed is the configuration file that the user may modify. Several fields in the configuration can be modified, one main field being `CONF_MAX_ENTRIES` which defines the maximum number of log entries that can be stored in the buffer.

5.6 Interface

Table. 7: Interface for C&C View

Elements	Description
Interface Identity	LogInterface. LogInterface shows detailed information about various types of system log events over different time range.
Resources Provided	Pre-Conditions: max-entries. REQUIRED. stored number of entries (Default value:50) fire_event. REQUIRED. whether events are fired (Default value: false) Post-Conditions: System_event_log returns all following information in data types in JSON format.
Data Types	1. {String} level: WARNING or ERROR depending on severity 2. {String} source: the file that provoke the error 3. {String} exception: full stack trace if available, an empty string otherwise 4. {String} message: descriptive message of the error 5. {String} timestamp: the time of the occurrence of the events
Error Handling	1. 500 Server Error: Internal Server Error ("can not get logs from container which is dead or marked for removal") The error occurs when system could not get logs 2. ERROR (MainThread) [homeassistant.bootstrap] Unable to setup error log /config/home-assistant.log (access denied) The error occurs when system tries to setup the error log 3. ERROR (MainThread) [homeassistant.loader] Error loading homeassistant.components.variable. The error occurs when system tries to find the installed dependencies
Quality Attribute	Reliability : The interface ensures the quality attribute and enables to access to the information about all logged errors and warning messages from Home Assistant. Only 50 last errors and warnings can be shown in the interface. Users can change number of logs with the help of max_entries parameter.
Rationale	The LogInterface allows users keeps track of log data and make it simple to differentiate different types of logs such as warning,error or fatal. The users can filter the system log based on their requirements, by changing the filters.

<table>
<tr><th>Elements</th><th>Description</th></tr>
<tr><td>Usage Guide</td><td>The LogInterface gives the user’s ability to easily see and configure the log files. Examples of the log events can also be found in Home Assistant log file (home-assistant.log). An example of the log file:



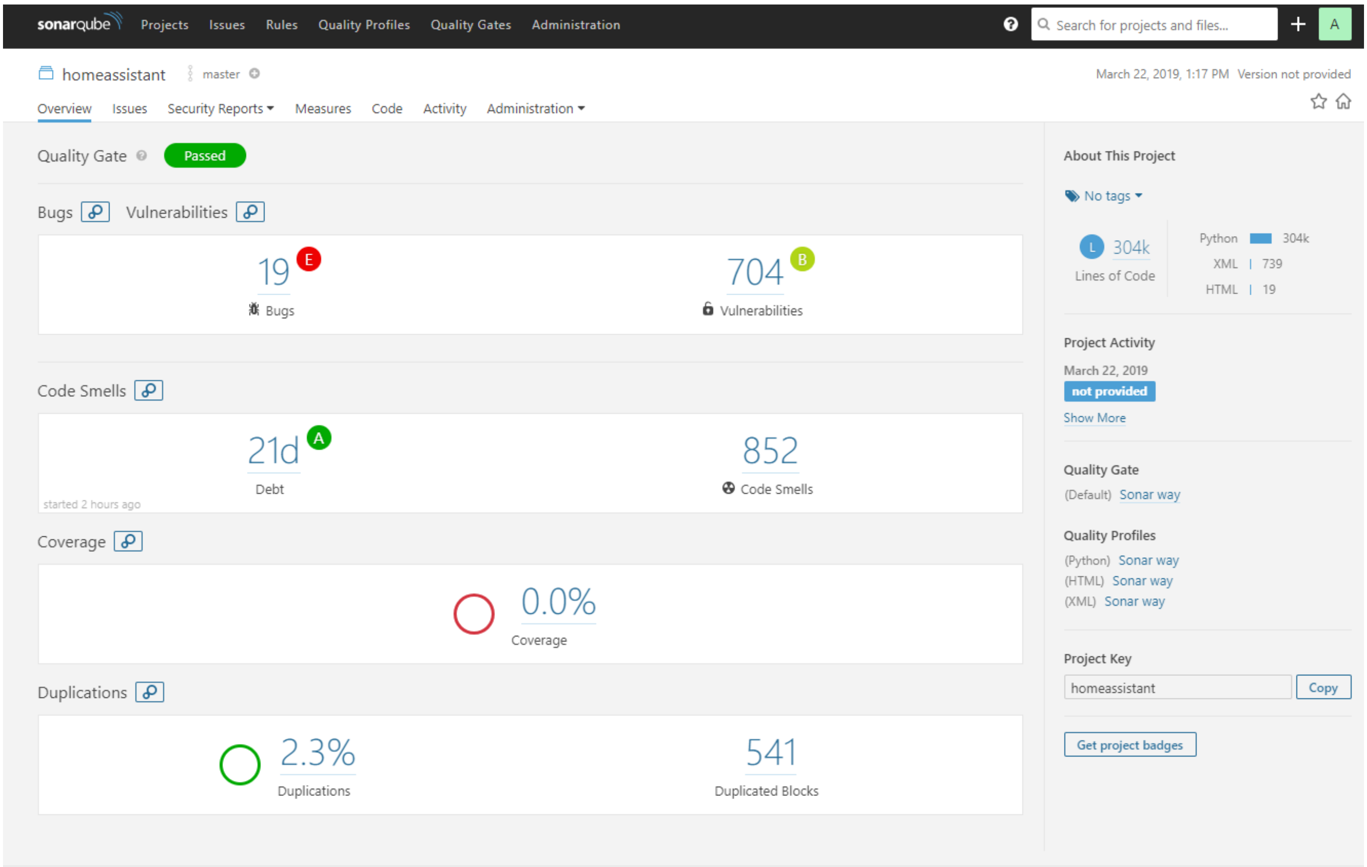
```
2019-02-14 16:20:35 ERROR (MainThread) [homeassistant.loader] Unable to
find component system_healt
2019-02-14 16:20:36 ERROR (MainThread)
[homeassistant.components.device_tracker] Error setting up platform
google_maps
Traceback (most recent call last):
  File "/home/fab/Documents/repos/ha/home-
assistant/homeassistant/components/device_tracker/__init__.py", line 184,
in
[...]
```
</td></tr>
</table>

6.0 Code Quality & Technical Debt Analysis

Herein resides a report on our analysis of Home Assistant’s codebase to assess software quality and technical debt. To conduct our analysis, we first, utilizing the tool SonarQube, identified and analyzed metrics pertinent to software quality. Second, we surveyed the issue tracking section of the Home Assistant GitHub repository to discern and discuss two design tradeoffs that could potentially have long term consequences. In our analysis, we use a local instance of SonarQube, but a instance of the prokect can be found on SonarCloud at this link.

6.1 SonarQube

SonarQube is a critically acclaimed, robust, static analysis programming tool that can quickly scan large codebases, detecting bugs, code smells, and security vulnerabilities on over 20 programming languages [1]. Figure. 8 is an overview of the results from a scan of Home Assistant’s codebase using SonarQube.



homeassistant master

March 22, 2019, 1:17 PM Version not provided

Quality Gate Passed

Bugs Vulnerabilities

19 E Bugs

704 B Vulnerabilities

Code Smells

21d A Debt

852 Code Smells

started 2 hours ago

Coverage

0.0% Coverage

Duplications

2.3% Duplications

541 Duplicated Blocks

About This Project

No tags

304k Lines of Code

Python	304k
XML	739
HTML	19

Project Activity

March 22, 2019

not provided

Show More

Quality Gate

(Default) Sonar way

Quality Profiles

(Python) Sonar way
(HTML) Sonar way
(XML) Sonar way

Project Key

homeassistant Copy

Get project badges

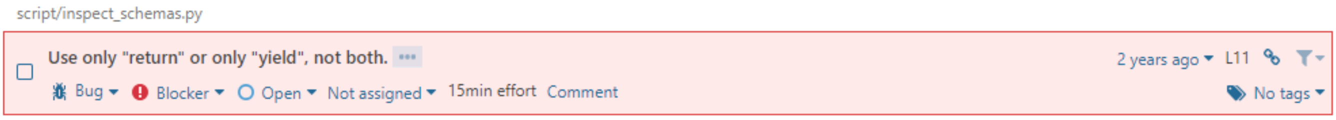
Figure. 8: Project Overview

As can be seen in the top corner of the figure, Home Assistant passed SonaQube's quality gate test and is ready for production. So, even though SonarQube identified 19 bugs, 704 vulnerabilities, 21 days of technical debt, 852 code smells, 0.0% coverage, and 2.3% duplications with 541 duplicated blocks, the impact of these metrics to software quality is nearly negligible to the overall function of the system. The following sections will elaborate on these results and their impact on software quality.

Bugs

SonarQube identified 19 bugs in the Home Assistant codebase. None of which are flagged as issues in the issues section of the Home Assistant GitHub respository, meaning they are not bugs, but improper python syntax used to make Home Assistant function. These bugs have three distinct types. The sections below describe each bug type: they provide an example of noncompliant code that caused it and they discuss the reality of the bug's severity. Having bugs rated blocker, SonarQube gave Home Assistant a reliability rating E. This is not accurate as the bugs identified by SonarQube are not that severe.

Type 1


script/inspect_schemas.py

□ Use only "return" or only "yield", not both. ···

Bug · Blocker · Open · Not assigned · 15min effort · Comment

2 years ago · L11 · No tags

Figure. 9: Type 1 Bug Flag

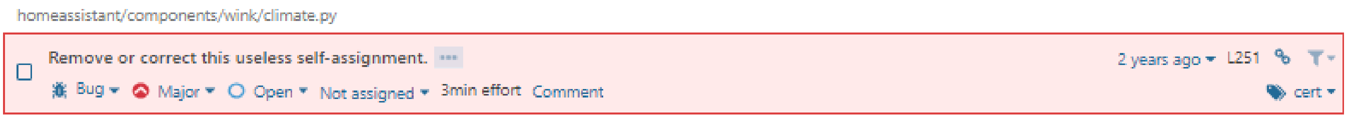
Noncompliant Code Example

```
def explore_module(package):

    """Explore the modules."""
    module = importlib.import_module(package)
    if not hasattr(module, '__path__'):
        return []
    for _, name, _ in pkgutil.iter_modules(module.__path__, package +
'.'):
        yield name
```

Using both yield and return in a function can result in a syntax error, but it is cautiously used by Home Assistant to achieve functionality. Therefore, is not a blocker bug like it was rated.

Type 2



homeassistant/components/wink/climate.py

Remove or correct this useless self-assignment.

Figure. 10: Type 2 Bug Flag

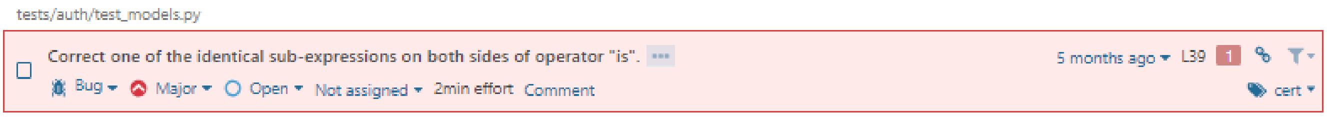
Non-compliant Code Example

```
if target_temp_high is not None:
  target_temp_high = target_temp_high
```

Discussion

These self assignment bugs are simply workarounds to ensure proper functionality of certain components. Therefore, it is not a blocker bug like it was rated.

Type 3



tests/auth/test_models.py

Correct one of the identical sub-expressions on both sides of operator "is".

Figure. 11: Type 3 Bug Flag

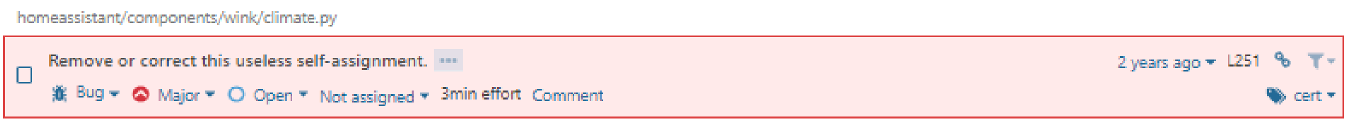
Noncompliant Code Example

```
assert user.permissions is user.permissions
```

These assertions are used to do testing for Home Assistant. Therefore, they are not major bugs like they were rated.

Vulnerabilities

SonarQube found 704 vulnerabilities in Home Assistants codebase. All 704 vulnerabilities flagged by SonaQube are the same. Each of them are of the form shown in Figure. 12.



homeassistant/components/wink/climate.py

□ Remove or correct this useless self-assignment. ···

Bug ▾ Major ▾ ○ Open ▾ Not assigned ▾ 3min effort Comment

2 years ago ▾ L251

cert ▾

Figure. 12: Vulnerability Example

Vulnerable Code Example

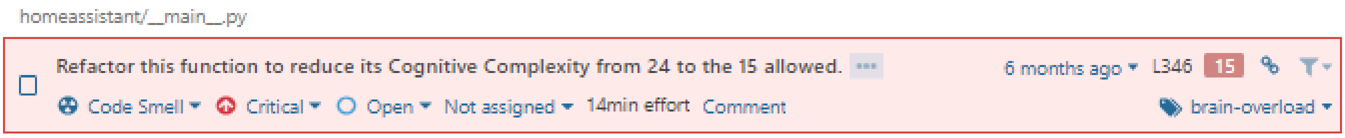
```
AXIS_DEFAULT_HOST = '192.168.0.90'
```

These vulnerabilities are a result of IP addresses being hardcoded throughout Home Assistant's codebase. These vulnerabilities result in real technical debt, which we discuss in the technical debt section of our analysis.

Code Smells

SonarQube found 826 code smells in Home Assistant's codebase. There are too many to represent all the different types in this report, but there were three types that comprised a majority of the smells.

Type 1



homeassistant/__main__.py

□ Refactor this function to reduce its Cognitive Complexity from 24 to the 15 allowed. ···

Code Smell ▾ Critical ▾ ○ Open ▾ Not assigned ▾ 14min effort Comment

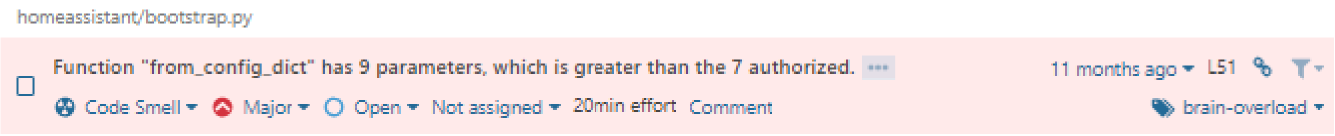
6 months ago ▾ L346 15

brain-overload ▾

Figure. 13: Type 1 Smell Flag

Cognitive Complexity is a rating of how hard it is to understand the code's control flow. [2] These smells make up a large portion of Home Assistant's code smells. Making methods have a lower cognitive complexity will increase readibility and make it easier for others to contribute to Home Assistant.

Type 2



homeassistant/bootstrap.py

□ Function "from_config_dict" has 9 parameters, which is greater than the 7 authorized. ···

Code Smell ▾ Major ▾ ○ Open ▾ Not assigned ▾ 20min effort Comment

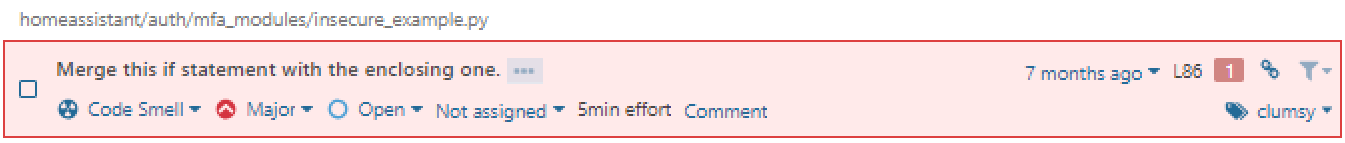
11 months ago ▾ L51

brain-overload ▾

Figure. 14: Type 2 Smell Flag

This is another smell that comprises a large portion of Home Assistant's code smells. Having fewer parameters would increase readability and type safety of the code.

Type 3



homeassistant/auth/mfa_modules/insecure_example.py

□ Merge this if statement with the enclosing one. ··· 7 months ago ▾ L86 1

Code Smell ▾ Major ▾ Open ▾ Not assigned ▾ 5min effort Comment clumsy ▾

Figure. 15: Type 3 Smell Flag

This smell results when an ''if'' statement is nested when it could be merged with the enclosing one. This is unnecessary and adds additional lines of code and complexity to Home Assistant

Technical Debt

SonarQube found that Home Assistant has 21 days worth of technical debt with a debt ratio of 0.1% . 21 days is the man-hours that SonarQube estimates it would take to correct all the codesmells it found. The debt ratio is the ratio between the cost to develop the software and the cost of fixing the code smells. Given that Home Assistants codebase is so large, the debt resulting from the code smells is very small in comparison to the amount of time spent developing it. Having a debt ratio < 5% Home Assistant was given an A for maintainability.

Coverage

Coverage is a mix of Line Coverage and Condition Coverage. Line Coverage answers the question: Has this line of code been executed during the execution of unit tests? Condition Coverage answers the question: Has each boolean expression been evaluated to both true and false? [2] SonarQube got a resulting coverage of 0%. This is not accurate, as Home Assistant has a `tests` module with many unit tests.

Duplication

SonarQube found that only 2.3% of Home Assistant's lines are duplicates. Most of these duplicates are from unit tests. Refactoring these duplications could improve compilation and testing time.

6.2 Technical Debt in issue tracker and source code

Not Upgrading Z-wave

Technical debt occurs when a design or construction approach is taken that's expedient in the short term, but that creates a technical context that increases complexity and cost in the long term. Currently, home assistant's current integration is using Z-Wave while there is a newer version released called OpenZWave. Z-Wave is a mesh network, where each node knows its surrounding nodes and can send packets through them. Using routing allows the system to successfully overcome obstacles between nodes that prevent them from communicating directly. There are some visible issues that Z-wave causes as when the number of devices for the system becomes large, users have reported large delays in the network. These issues have been reported on GitHub issues and on the home assistant forums, which the cost of fixing each issue would place a significant burden on developers. Replacing Z-Wave with OpenZWave implemented by SiLabs public SDK could alleviate the cost of maintaining the network in HomeAssistant as OpenZWave is upgraded to be more reliable, and is actively maintained. Not replacing Z-Wave would accrue debt interest of having more complicated code to handle the Z-Wave problems, resulting in longer release times, and longer test runs.

Hardcoded IP Addresses

Looking into the code, we noticed that the private network allocations are hardcoded in `/homeassistant/util/network.py`. This network utility provides functionality for whitelisting local networks by checking if the IP address is present in a predefined list of private addresses. Hardcoding an IP address into source code is a bad idea [3] and can raise the following Technical Debts:

- A recompile is required if the address changes
- It forces the same address to be used in every environment (Dev, Sys, QA, Prod)
- It places the responsibility of setting the value to use in production on the shoulders of the developer
- It allows attackers to decompile the code and thereby discover a potentially sensitive address

The list of IP addresses could be easily configured in a separate configuration file that could be added to .gitignore to ensure secrecy of any sensitive address. The configuration file could then be easily changed as needed to use in different environments.

7.0 Pull request for Multifactor Authentication workflow diagram

Since we have examined the architecture and documented it for this past 3-4 months, we wanted to contribute to their developer documentation. Their documentation can be found here and the repository here. In their documentation, there was a clear `TODO` to draw a diagram for the Multi Factor Authentication Workflow. We examined the architecture of Multifactor Authentication and built a diagram of the workflow. Fig. 16 shows the workflow diagram.

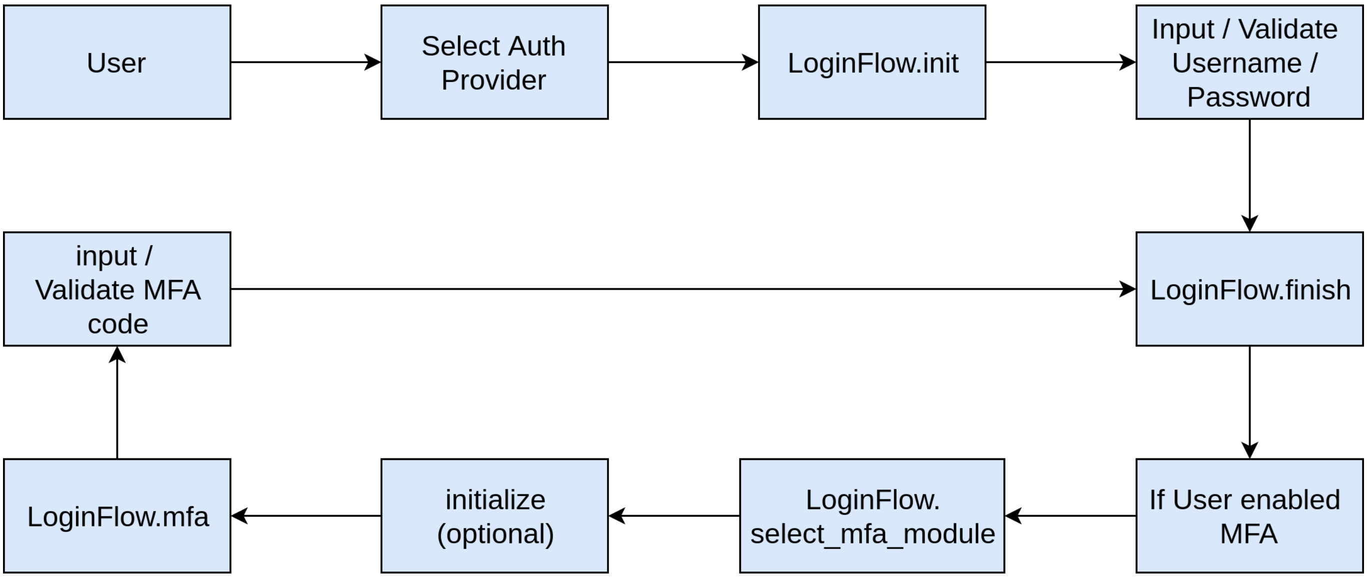


Figure. 16

Good news is the pull request was reviewed by the founder of Home Assistant and accepted. He asked for the link of the image for any future work that could be needed and was very nice to thank us for contribution.

Link of the pull request: Multi Factor Authentication Workflow Diagram Pull Request

8.0 Conclusion

Through assessing Home Assistant's business orientation and then determining their ASR's we were able to identify the relevant QAS's that are fulfilled by the Home Assistant platform. We used these QAS's as a guide to create both a module and C&C view--architectural documentation that we hope prove useful to users of the system and, more specifically, the relevant modules. We then conducted a code quality and technical debt analysis using the static analysis tool SonarQube. Using this tool and knowledge we gathered by reading community posts and the issues section of Home Assistant's GitHub repository, we found two issues that have incurred technical debt for Home Assistant. However, although our analysis found a couple instances of incurred technical debt, it largely revealed how well-developed Home Assistant is. It was great to work on this project and we were amazed by their platform and its architecture.

References

1. SonaQube Wiki
2. Metric Definitions
3. CERT, MSC03-J. - Never hard code sensitive information