Group5 Report

ANDREW SOWINSKI,
HAOCHENG SONG,
QIHAO WANG,
ZIWEI ZHOU

This report introduces the architecture, how it works, and used open source of the IDE.

Link of Github: https://github.com/UF-MP-Group-5/Part1/tree/main

1 ARCHITECTURE

There are two basic steps for IDE, first step is “initiation phase”. In this phase, every Pi should connect to vss and running Atlas. Then IDE’ backend will receive descriptive tweets from all Pi that share same space_id. Backend will store these metadata into temporary memory, and sending all these information to frontend, so that frontend is able to show these metadata about our space. User can look through what services and relationships we have and build some funny applications base on our services. At this phase, local application(json file) will also be loaded into backend memory and frontend webpage to show.

Second phase is “running phase”. User can interact with frontend webpage to select services and build applications, running applications with other special function like saving apps, upload apps . Frontend will keep listening on user interaction, and sending requestion as JSON to backend. Backend has many controller to deal with different types of requests, and trigger specific functions. If functions are Pi related, like running applications, then backend will calling services on Pi by using Atlas. Backend also able to get possible result back from Pi, and return result to frontend to show.

Running phase will keep running unless some Atlas-related bug appears, normal bugs will not crash running phase.

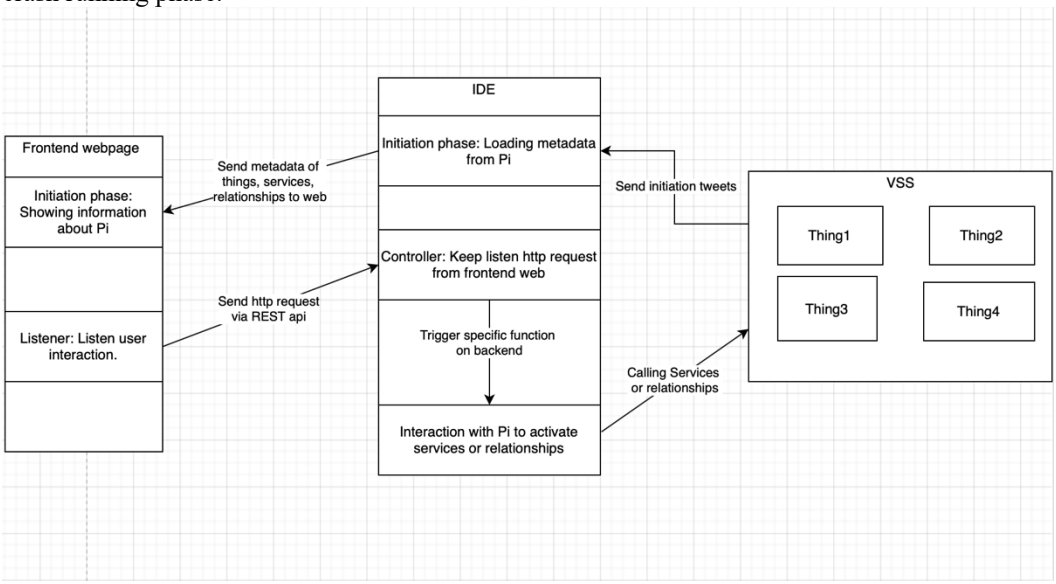

Fig. 1. Architecture of the IDE

2 SCREENSHOTS OF TABS AND FUNCTIONALITIES

2.0 Navigator

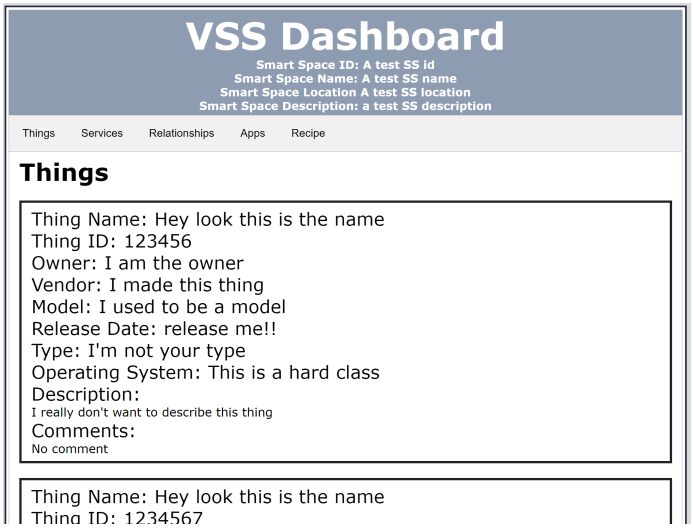

Fig. 2. Navigator

2.1 Thing Tab

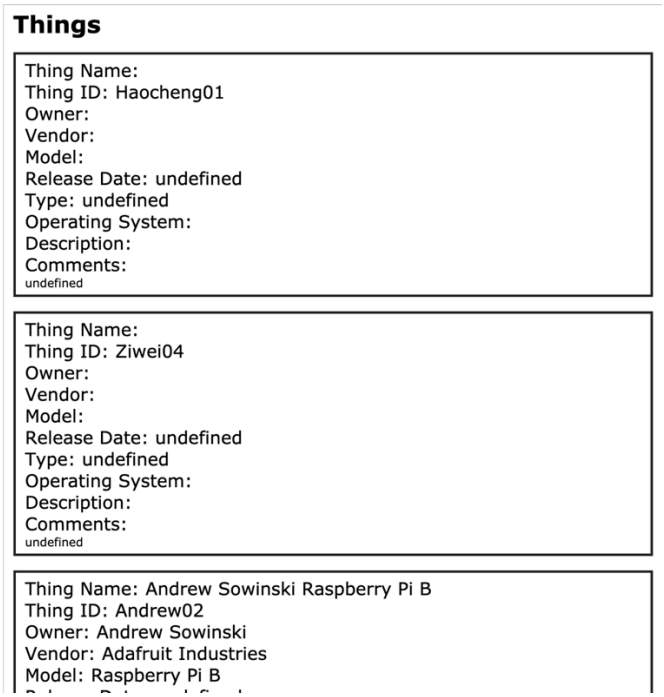

Fig. 3. Thing tab

2.2 Service Tab

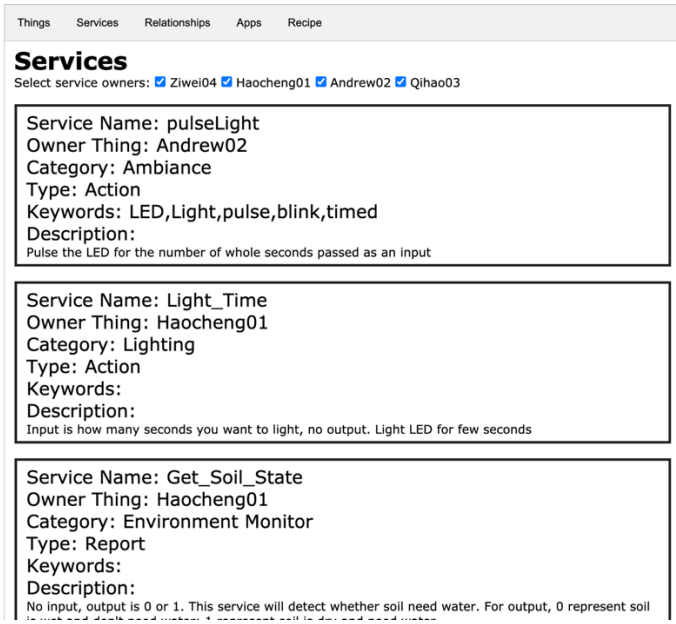

Fig. 4. Service tab: Showing available services can be grabbed to recipe tab

2.3 Relationship Tab

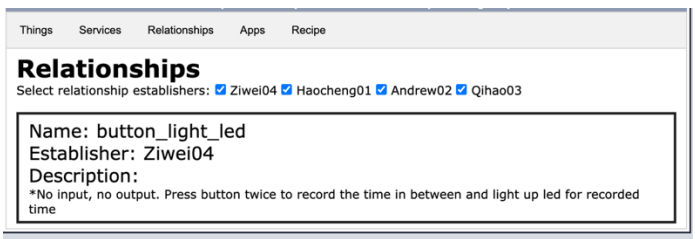

Fig. 5. Relationship tab: Assembled by services with condition

2.4 Recipe Tab

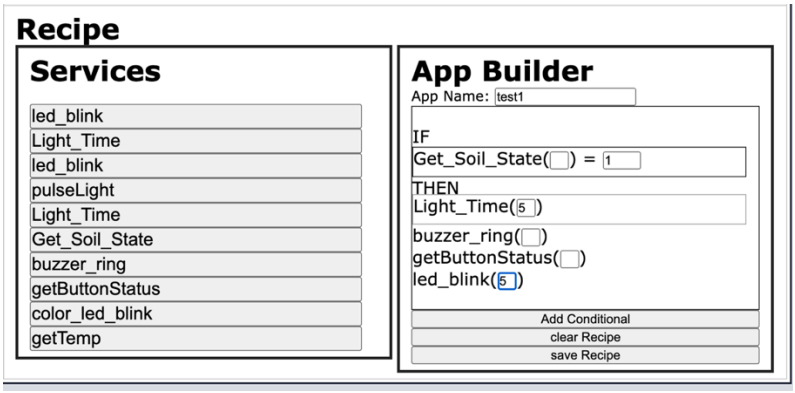

Fig. 6. Recipe tab: Grab services to build app

2.5 App Tab

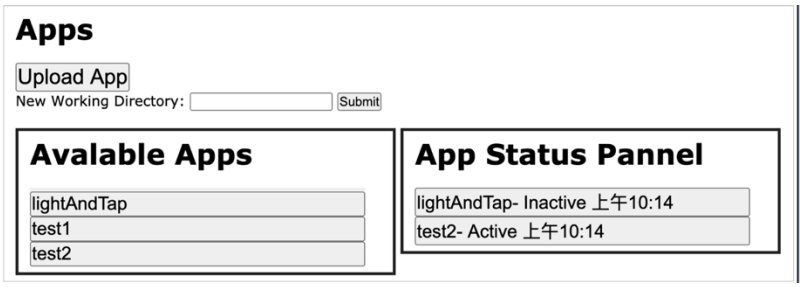

Fig. 7. App tab: Press available app to get into manage app

2.6 Application manager

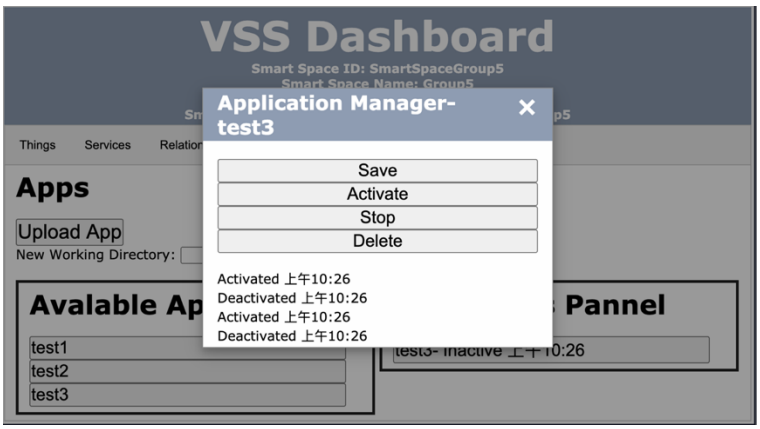

Fig.8. application manager: save to local; activate service; interrupt services which haven't been executed in a sequence of app; delete app from available apps

3 FRONT END DOCUMENTATION

Below are descriptions of each front-end function and its parameters. Some are visual in nature, some are used for front-end to front-end communication, and some are used for front-end to back-end communication.

function addThing(ID, name, owner, vendor, model, release, type, OS, description, comments): Used to add a Thing to the "Things" tab of the UI.

function addService(name, category, type, keywords, description, owner): Used to add a Service to the "Service" tab of the UI.

function addRelationship(parent, child, description): Used to add a Relationship to the "Relationships" tab of the UI.

function addAvailApp(name, data): Adds an app unit to the available apps menu.

function addStatusApp(name, data): Adds an app to the status panel as "active".

function activateApp(): Activates an inactive app.

function stopApp(): Deactivates an active app.

function stopAppFromBackend(appname): Allows the back end to update the UI to reflect that an app is no longer running.

function updateHistory(historyJson): Updates the app history shown on the application manager.

function saveApp(): Saves an app to the working directory as a json through communication with the back end.

function deleteApp(): Deletes an app from the UI and from the working directory (using the back end).

function filterRel(): Filters relationships based on selected parents.

function filterServ(): Filters services based on selected owner device.

function onLoadFunc(): Does UI setup on load.

function setSmartSpaceInfo(ssID, ssName, ssLoc, ssDescrip): Updates the smart space information in the header.

function newWDSubmit(): Sends the new working directory to the back end.

function allowDrop(ev): Drag and drop support function.

function drag(ev): Drag and drop support function.

function drop(ev): Drag and drop support function for Recipe tab.

function drop2(ev): Drag and drop support function for Services tab.

function addConditional(): Adds a conditional block to the Recipe builder.

function clearRecipe(): Clears the recipe builder.

function saveRecipe(): Saves a recipe as an app to the Apps tab.

function makeModal(caller): Displays the modal pop-up window.

function removeOldApps(): Removes apps from the App status window that have been inactive for 5 or more minutes

function oneSecondCycle(): Used to check for updates once per second

function doClickAction(tempThis): Click and double-click support function for Apps tab.

function doDoubleClickAction(tempThis): Click and double-click support function for Apps tab.

function handleClick(tempThis): Click and double-click support function for Apps tab.

function handleDoubleClick(tempThis): Click and double-click support function for Apps tab.

function openTab(evt, tabName): Opens a specific tab based on the parameters.

4 BACK END DOCUMENTATION

Below are the documentations of backend's functions, I will introduce them in types of entity classes, controllers, and util mehthods.

4.1 Entity class

Service: It's basically mapping to Service from our Pi, it include all metadata of service in pi, include name, thing_id, entity_id, space_id, vendor, api, type, appcategory, description, keywords Pi. Service class has important method **public String call(String inputs)** which will activate this service via socket to Pi and return result.

Relationship: It's basically the relationship of Pi, contains include type of relationships, first service, second service and description. It has method **public String call(String inputs)** which will activate its compontent services by specific order.

Thing: This class represent Things of Our Pi, it include thing_id, space_id, name, model, vendor, description, owner, OS, also with ip address and port number of this thing.

App: This class represent App, which is the combination of services. It contains App's filename, appName, workingDirectory, units(Which is an array of another class) and originJson(Which is the originally json of this app). There is an inner class Unit represent every single unit of App. **public String[] run()** will run the application's unit one by one, and return each result; **public void deleteApp()** will delete current application from local file; **public void saveApp()** will save current application to working directory.

AppForWeb: It's a App class for frontend to show, only include appName and originJson.

4.2 Controller

public Thing[] initiationOfThings(): Listen on GET method on uri "/getThing", will initiate all things in to frontend.

public Service[] initiationOfServices(): Listen on GET method on uri "/getService", will initiate all services in to frontend.

public AppForWeb[] initiationOfApps(): Listen on GET method on uri "/getApps", will initiate apps from local file.

public Relationship[] initiationOfRelationships(): Listen on GET method on uri "/getRelationship", will initiate relationships to frontend.

public void activateApp(@RequestBody String request): Listen on POST method on uri "/activateApp", will activate a specific app.

public void saveApp(@RequestBody String request): Listen on POST method on uri "/saveApp", will save a specific app.

public void deleteApp(@RequestBody String request): Listen on POST method on uri "/deleteApp", will delete a app from local file if possible.

public void changeDirectory (@RequestBody String request): Listen on POST method on uri "/ changeDirectory", will change default saving and working directory for apps.

4.3 Some Util functions

public static void getTweetsFromPi(): Get tweets with select space_id from Pis, and loaded their metadate into memory.

public static void scan(): Scan working directory to load possible apps

public static boolean setWorkingDirectory(String path): Set new working directory instead of default.

5 OPEN SOURCE USED

Spring Boot.
It is one of the most popular open source projects in the cs domain. It is mainly for development of Java backend of web application with support of many modules such as spring data.
The usage for our project is to build endpoints for front-end so that front-end can send requests to run real functionalities.

6 CONTRIBUTION

Andrew Sowinski(30%): Design whole frontend webpage, build interaction components that user can use to interact with IDE to build their apps.

Haocheng Song(30%): Design frame of backend, achieve interaction between frontend and backend, achieve interaction between backend and Pi.

Ziwei Zhou(20%): Achieve reading descriptive tweets from Pi, and mapping these metadata into Java class; Test project's functionality.

Qihao Wang(20%): Achieve special function on backend(save application, delete appcliation…); Test project's functionality.