University of Puerto Rico
Mayaguez Campus
Engineering Faculty

Visual Server

Report

Team Members:

Amanda J. Vázquez: amanda.vazquez2@upr.edu

Javier A. Ortiz: javier.ortiz13@upr.edu

Luis Rivera Marquez: luis.rivera79@upr.edu

Programming Languages
Prof. Wilson Rivera

I. Introduction

What is IoT? The internet of things, or IoT, is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction. Now imagine yourself on the verge of creating your first device. The simplest way is to create a small server. How complicated can this be? It depends really. Nothing is tough, it totally depends on your source and the way you are learning things.

What do we bring the table? Introducing Visual Server (ViSe), a programming language for servers. With this language you can easily and simply start a server, setup the routes and begin sending or receiving requests. Our motivation behind this language is to lower the entry barrier into the IoT industry by making simplicity our greatest weapon.

From the creation of severs to the assignation of routes, it is all done in less than 10 lines of code. How do we do this ? We only worry about send and receive operations to your server. These two operations are essentially the key for anyone to begin working on any IoT device. That is the beauty of ViSe. Our goal is that one day, developers look at our language as their first choice for starting IoT products and say that ViSe is the future of “do it yourself” IoT devices.

II. Language Tutorial

A. Installation

1. You need Python 3.7 or higher.
2. Download the GitHub repository at: https://github.com/AmandaJanice/ViSe
3. Open terminal and travel to the folder of the project
4. Run: `pip install -r requirements.txt`

B. Using Visual Server

1. To start using ViSe navigate to the src folder of the project on the terminal and run: `python3 ViSe.py`
2. Creating your own local server:
   - ➔ Create a server instance and store it on a variable with :

     ```
     serverInstance = createServer(port = long);
     ```

   Note: You can leave the parameter out, it will create the server on the port 80.
3. Start your server: `serverInstance : start;`

4. Storing data, you can only create json objects or integers:
   - ➔ For json objects

     ```
     variable = json:{ string : string};
     ```

     You can enter more than one data, example

     ```
     variable = json:{ string:string, string:string, ...};
     ```

   - ➔ For integers

     ```
     variable = 68;
     ```

5. Setting your route instance:

   ```
   routeInstance = serverInstance : setRoutes(url = string);
   ```

6. Setting action to your route instance:
   - ➔ Reading data on your server

     ```
     routeInstance : readData(body = variable);
     ```

   - ➔ Creating data on your server

     ```
     routeInstance : createData(object = variable);
     ```

   Note: You can not set more than one action to your route instance.
7. Communicating with external open servers:
   - ➔ Getting and storing a json object of the external data

     ```
     variable = httpGet(url = string);
     ```

     We have found, till the creation of this document, only two links:

     1. "https://reqres.in/api/users?page=2"
     2. "https://jsonplaceholder.typicode.com/posts"

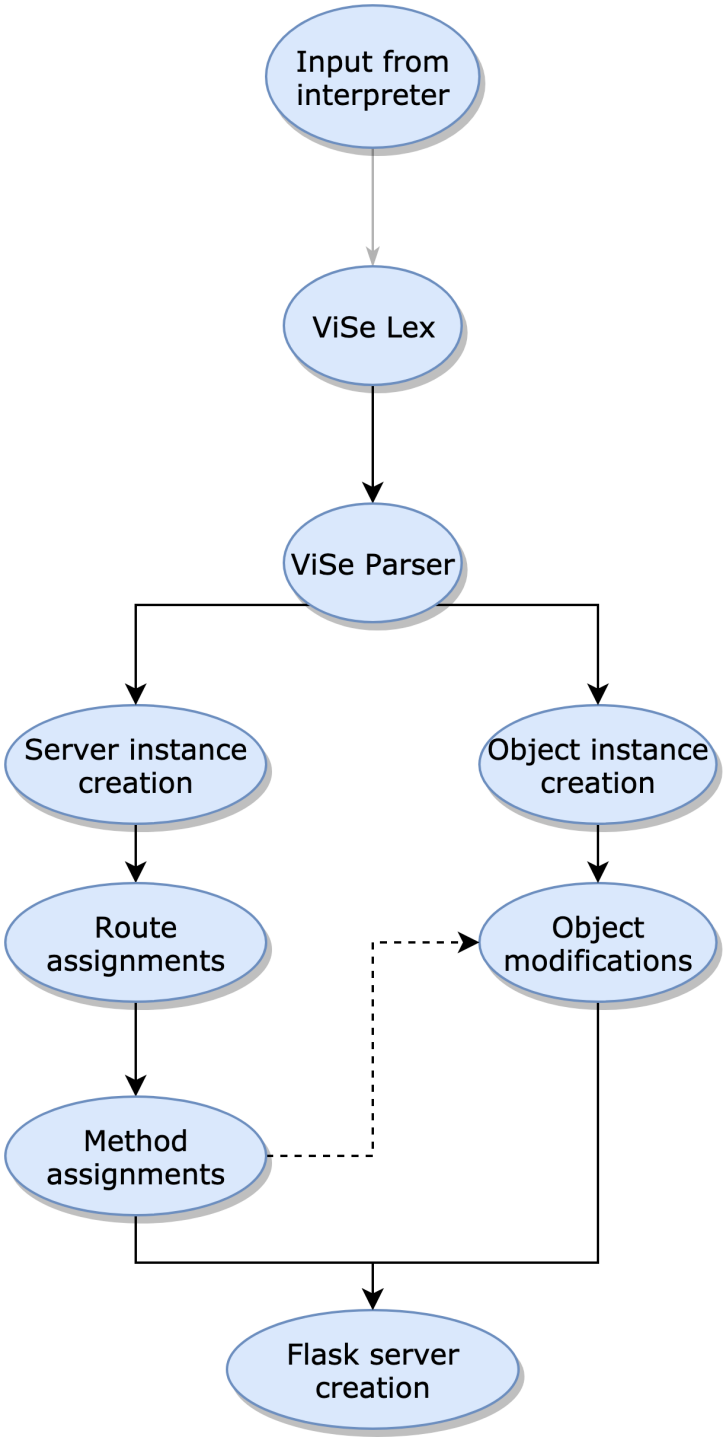
III. Language Reference Manual

API:

1. ***createServer (port= 3000)*** : Initializes the server and assigns port value.
2. ***json : {string:string, ...}*** : creates a JSON object.
3. ***serverInstance : setRoutes(url = string)*** : creates your route instance
4. ***routeInstance : createData(object= data)*** : creates object and assigns data into that object.
5. ***routeInstance: readData(body= data)*** : set action to route.
6. ***serverInstance : start*** : start running your server
7. ***test = httpGet(from = string)*** : stores post request.

IV. Language Development

A. Translator Architecture



B. Describe the interfaces between the modules

1. Input from Interpreter
    a) User input received on the interpreter that is then executed
2. ViSe Lex
    a) Lexer to verify correctness of syntax. uses PLY Lex

3. ViSe Parser
    a) Parses lexed text into executable functions. uses PLY Yacc.
4. Server Instance Creation
    a) Creates Flask server instance that can be started at a specified port. Other attributes, like routes, are added to this instance later.
5. Route assignments
    a) routes are assigned to the server instance to be accessed by any clients that want to connect to the server
6. Method Assignments
    a) Assignment of specific method (createData or readData) to a route. These methods are similar to POST and GET respectively. We also have httpGet which submits a GET request to any server.
7. Object Instance creation
    a) objects store json data, as well as routes
8. Object Modifications
    a) Assigned methods can modify object instances. A createData method can rewrite an instance of an object.
9. Flask server creation
    a) Start running flask server. After this the server is live at the port chosen when creating the server instance.

C. **Describe the software development environment used to create the Translator**.

1. Python
    a) Programming language used to create the interpreter for ViSe
2. Flask
    a) Python library that allows the creation of servers. Used to implement server functionality
3. Github
    a) Version control system used throughout development.
4. PLY
    a) Python Library used to create parser and lexer
5. Text editors
    a) Used atom and PyCharm. When using atom dependencies were installed in a Python virtual environment. PyCharm takes care this for us

D. Describe the test methodology used during development.

1. To test the language we began by making each small component individually. First the intermediate code and the lexer. Code a part, test it, debug it. After the lexer was done, the parser was made using the same methodology, after making sure all the tested inputs worked. We began integration. When integrating a similar approach was used for testing, integrating each functionality into the parser one by one and checking for bugs, and fixing them appropriately.

E. Show programs used to test your translator.

1. Terminal (Command prompt on Windows)
    a) used for running the interpreter and checking for error outputs
2. Postman
    a) Used to test requests for sending or receiving data for the server
3. Google Chrome (any browser should work)
    a) Used for testing route outputs

V. Conclusions

ViSe is a simple way to learn programming compiled using python and ply. When we started working on this project we only had one goal, and it was to create a basic syntax language for servers so that users can have a user-friendly experience when trying our language. We also wanted to illustrate the functionality of how a server is created and can be easily implemented.

While developing ViSe we could really tell apart all the advantages this language had to offer. It was like making an update from JavaScript to Typescript, it is a perfect example of how a high level programming language works, and how it can facilitate complex task from new developers without dealing with the complicated concepts of Internet of Things.