2025, 29(1), 4–12. Received 10 March 2025. Accepted 24 March 2025.

SOLUTION OF REMOTE MANAGEMENT FOR COMPUTER NETWORK INFRASTRUCTURE USING INTEGRATED NETBOX

Liudvikas Kaklauskas
Šiaulių valstybinė kolegija / Higher Education Institution
Rimantas Pilypas
Šiaulių valstybinė kolegija / Higher Education Institution

Annotation

The purpose of the research is to create an automated switch configuration management system using the NetBox platform. A Python program developed for data processing integrated into NetBox. System objects were be linked by Django relational relationships using REST API methods. An interaction model of system components was implemented using SSH Paramiko and Netmiko libraries. The integration points of NetBox platform REST API were supplemented with seven switch port configuration methods using Token authentication. Testing of the system showed that the port parameters data was successfully transferred in an average of 135 milliseconds. By research established that SSH protocol library Netmiko is better compatible with different switch models.

Keywords: NetBox, Django, Rest API, *SSH*, *Netmiko*

Introduction

The manual administration of computer network infrastructure using properly structured and documented data ensures smooth service provision. Rapidly changing business conditions and constantly growing technological requirements encourage the search for more efficient solutions that help to quickly and accurately manage increasingly complex computer network systems. For the dynamic provision of today's services, it is possible to automate network equipment configuration changes (Nketsiah et al., 2022), apply specialized management tools (Dzemydienė et al., 2023), use network management and decision-making systems (Zhu, 2021). The need for change also arises because updating device configurations manually inevitably leads to unforeseen errors and a lot of time is lost. L. A. Haibeh et al. (2022) analyse 5G communication management automation solutions. W. Irtaza (2021) recommends using the open-source automation tool Ansible in his book, which is suitable for configuration management, plug-in installation, service organization, and infrastructure provision. M Priyadarsini and P. Bera (2021) recommend using a 3-layer software control model for computer network management. R. Lucifora (2024) analysed the possibilities of using the SuzieQ and NetBox platforms to manage network device configuration changes.

It should be noted that the analysed scientific literature examines general solutions for documenting and managing network infrastructure. It was found that solutions for network resource groups remote securely managing using the NetBox middleware with a network administrator-friendly interface have not been analysed.

The research goals are applying REST API and SSH libraries, create a system with the NetBox module for automated management of switch configurations.

Objectives: 1) Analyse the principles of switch documentation; 2) To design a NetBox module for automated switch configuration updates using REST API and SSH methods and integrate it into the system; 3) To test system functionality.

1. Infrastructure management model

Based on the analysis of similar solutions, it is planned that the NetBox middleware with an integrated switch cluster management component will be used for solution management, which will scan, verify and synchronize the switch port configuration according to the documented port data (Gupta, 2024, Uramova et al. 2024, Mulyana, Fakih, 2022). After analysing the NetBox user interface and scientific articles, it can be stated that the information area of the platform's home page is characterized by a clear layout of information, it is easily adapted to individual needs. The management area is also presented in a structured way, each management component is assigned to the appropriate group, which makes it easier to find the necessary objects (Lucifora, 2024).

The following functions have been realised in the system: documentation of port parameters on the NetBox platform; system administration using a dynamic command template library with the ability to view and them apply; DRF (Django REST framework) API integration points have been created through which requests can be executed using the REST framework. The system has an integrated error prevention mechanism that divides the sent command template into separate command lines and executes them one line at a time, thus preventing further command execution in case of an error. The

2025, 29(1), 4–12. Received 10 March 2025. Accepted 24 March 2025.

NetBox platform, on which the system is based, uses default values – the Python programming language, the framework Django , and PostgreSQL as the system database (Choi, 2024a, Šoška, 2021).

The tools selected for system development form a suitable initial environment for the development of the NetBox module project. The Python programming language, the Django framework based on it, and the PostgreSQL database are the default tools of the NetBox platform. The Visual Studio Code code editing tool and the integrated Git versioning system are the main platforms used to create module files and track project versions (Lin et al., 2024, bin Uzayr, 2022). The Postman API tool (Kore et al., 2022) and the GNS3 virtual device simulation platform were selected for query testing, they will help analyze API query responses and their execution time during the system plugin testing process (Šoška, 2021). Testing queries with Postman, responses are returned with additional information: a code of status (indicates whether the query response is successful or it is error message), the size of response, and the time it took for the response to be returned. The alternative API tool BDDFramework was rejected because it only returns the query response and does not provide more detailed response information (Chrisna et al., 2024). The Windows operating system was chosen for testing.

The SSH interface libraries Paramiko and Netmiko were selected for connecting to remote resources. Based on the functionality of the product being developed, these libraries are required to send command configuration sets to real physical network switches using the SSH protocol and are the most suitable choice considering the software's compatibility with other selected tools.

2. A model of computer network devices documenting

Research of the NetBox user interface showed that NetBox platform devices are divided into three main groups: devices - network infrastructure physical or virtual objects; device types - devices technical parameters; components of device - elements of devices. In order to add a device, it is necessary to create an organization in which the device is located, to specify its role and type. After properly documenting a device in the NetBox system, a component - a port can be added to it. The values of the elements of the newly created port can be assigned or selected from the drop-down menu. You can review and verify the entered data after documenting and filling components values. The framework of Django, NetBox REST API or customized methods of module can be used for programmatic management of data in the NetBox platform.

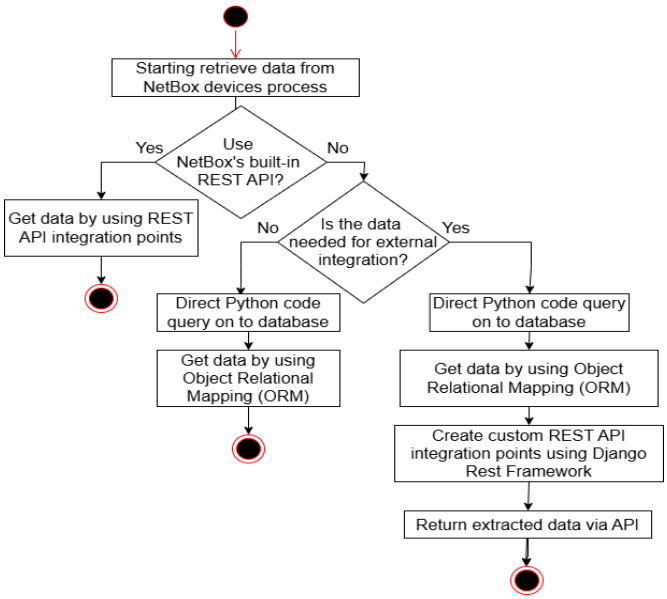

Fig. 1. *Diagram of data retrieval from NetBox devices*

The REST API method allows NetBox device data to be accessed through specially defined integration points. This method allows platform data to be accessed from other external systems, but the choice of integration points is limited. The data can be accessed directly from the NetBox database, because framework of Django uses an object-relational interface (Django, 2024). This solution is used only in the internal computer network and is more efficient in terms of performance, but external systems cannot access the data obtained using this method. Using object-relational queries directly to the NetBox database and the Python DRF library, device data is accessed through REST API method integration points, combining the functionality and external integration aspects of these methods. The

2025, 29(1), 4–12. Received 10 March 2025. Accepted 24 March 2025.

process of selecting a data retrieval method is illustrated in a UML (Unified Modeling Language) activity diagram (see Figure 1).

In the diagram we can see those different methods of data retrieval depend on the situation. The third method, the custom modules method, was chosen for the project implementation. The integrated Python module is extended with custom DRF integration points to access data through external systems. Object relational relationships in the module are used to directly retrieve information about ports from the NetBox database.

3. NetBox API project

The designed API component will be the main interface between the module users and internal processes. It will ensure the processing of requests and routing to the appropriate system layers. The functionality of the NetBox module API component is implemented through the Django REST Framework (DRF), which provides the ability to manage HTTP requests and associate them with appropriate functions in Python programming code (Jordon, 2019). API integration points are designed using URL, class-based views, and specific request handling functions (Dhadil et al., 2024). It should be noted that when using the request “GET /api/plugins/switch-status-updater/{server_place}/” instead of the variable “server_place” used in the request, specifying the name of the server device documented in the NetBox system, a JSON-format response is returned, which provides extended information about the switch port components (MAC, description(s), etc.). Requests "POST", "PUT", "PATCH" or "DELETE" with a relative URL path "/api/plugins/switch-status-updater/{server_place}/" initiate the configuration process according to the associated template of adjustable commands in the NetBox administrative part.

The server’s name "server_place" is used as a variable in HTTP requests and identifies the server device. The configuration to the switches is transferred via HTTP. REST API points of the NetBox system have been extended by using additional integrated module for thus reasons. Table 1 presents the relative URL paths of the designed module and their interfaces with components.

Table 1. NetBox module API integration points interface with program code methods and NetBox command templates

HTTP method	***URL relative path***	**Method of Python module**	**Template name**
GET	*,,/api/plugins/switch-status-updater/{server_place}/”*	*In ,,retrieve”* class *,,SwitchPortViewSet”*	Not applicable
GET	*,,/api/plugins/switch-status-updater/{server_place}/live”*	*In „retrieve_live_config”* class *,,SwitchPortViewSet”*	*,,{switch_model}_running_config”*
POST	*,,/api/plugins/switch-status-updater/{server_place}/”*	*In ,,setup_port”* class *,,SwitchPortViewSet”*	*,,{switch_model}_configure_port_post”*
PUT	*,,/api/plugins/switch-status-updater/{server_place}/”*	*In ,,setup_port”* class *,,SwitchPortViewSet”*	*,,{switch_model}_configure_port_put”*
PATCH	*,,/api/plugins/switch-status-updater/{server_place}/”*	*In ,,setup_port”* class *,,SwitchPortViewSet”*	*,,{switch_model}_configure_port_patch”*
PATCH	*,,/api/plugins/switch-status-updater/{server_place}/{template_name}/”*	*In ,,setup_port_with_template”* class *,,SwitchPortViewSet”*	*,,{template_name}”*
DELETE	*,,/api/plugins/switch-status-updater/{server_place}/”*	*In ,,setup_port“* class *,,SwitchPortViewSet”*	*,,{switch_model}_configure_port_delete”*

The table 2 describes all REST API methods and their purpose. Command templates are associated with switch models, because the syntax of the sent command set differs for each model (e.g. Mikrotik or Dell). The user describes the commands in the templates according to the syntax of a specific

2025, 29(1), 4–12. Received 10 March 2025. Accepted 24 March 2025.

model. The command template is not used in a "GET" request, when only the name of the server device is specified in the URL relative path, in this case the information is obtained directly from the switch port components on the NetBox platform in JSON format. Command templates are prepared for all other requests, because the module will connect to physical switches via SSH and execute commands to apply configuration methods or return parameter data.

Table 2. The purpose of REST API used methods in the module

Method	***URL* relative path**	**Purpose of the method**
GET	*/api/plugins/switch-status-updater/{server_place}/*	Getting port information from NetBox
GET	*/../../../{server_place}/live/*	Obtaining port information directly from the device
POST	*/../../../{server_place}/*	Applying the configuration to a device
PUT	*/../../../{server_place}/*	Applying the configuration to a device
PATCH	*/../../../{server_place}/*	Applying the configuration to a device
DELETE	*/../../../{server_place}/*	Applying the configuration to a device
PATCH	*/../../../{server_place}/{template_name}/*	Applying a configuration to a device using a selected template

The UML sequence diagram (see Figure 2) visualizes the main steps of a successful API request processing process. The diagram shows the sequence of actions of the API component for interaction with other components of the module. The process begins when the user sends an HTTP request with the corresponding URL address, using the “GET”, “PUT” and other methods. The API component receives the request and directs it to the corresponding function of the Python module via DRF. The method associates the request with a Jinja2 command template, based on which commands are generated ready for execution. The set of commands prepared for the switch is executed via the SSH protocol, using the Python Netmiko or Paramiko libraries (each template is pre-assigned with library selection values) (Choi, 2024b). After receiving the result from the switch, the response is processed and returned to the user in JSON format via the API component.

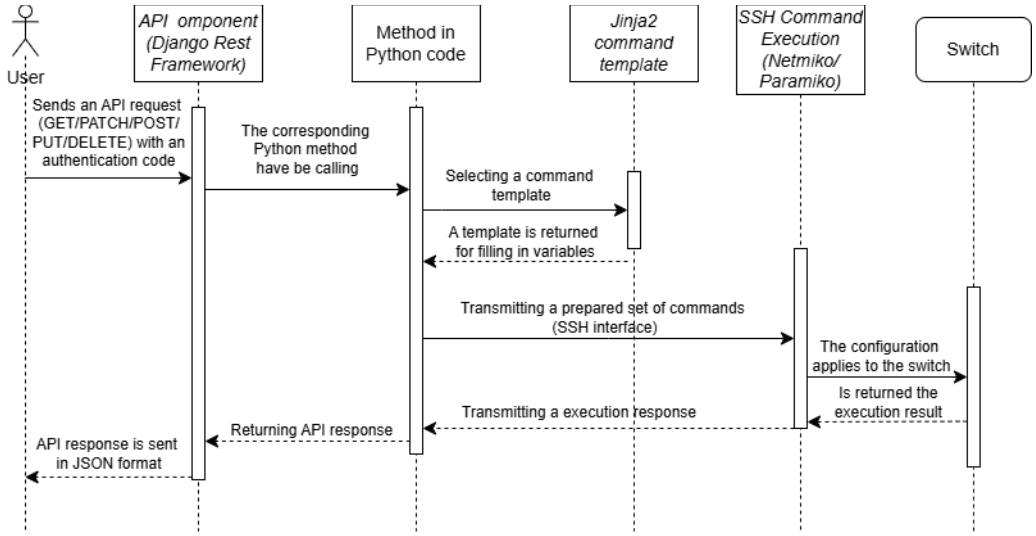

Fig 2. *UML sequence diagram of the process of successfully sending an API request*

This diagram shows the successful process flow when no error is encountered in the process. Message about the error return to the user in JSON format. API component implements 7 REST API integration points that ensure its functionality.

To ensure the functionality of the component, a designed database, which will store command templates, switch models, and command execution logs. A Django framework is uses in NetBox platform for creating data models which uses classes in Python. Each model corresponds to a PostgreSQL database table, and the model fields (class attributes) define the columns of the table (see Figure 3).

2025, 29(1), 4–12. Received 10 March 2025. Accepted 24 March 2025.

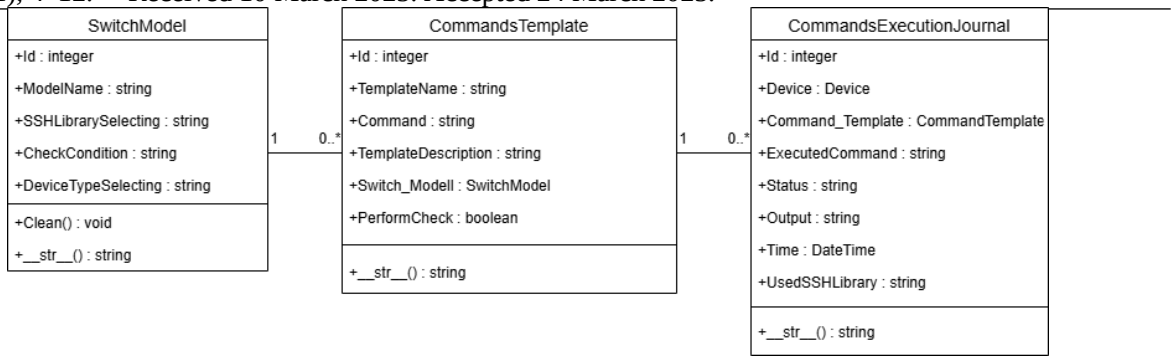

Fig 3. *UML class diagram of Django models required for the NetBox plugin*

Based on the established relationships between the command template and the switch model, it can be stated that one switch model can have zero or more associated command templates and each command execution log can be associated with only one command template. In Python programming code, the data model, in this case the command template, is described as a class. Based on these classes and using the Django ORM “makemigrations” and “migrate” commands, database tables can be automatically created. After properly describing the data models, database migrations can be performed and the corresponding tables are created in the PostgreSQL database. In Python programming files to write, read, update, and delete data it is logical to use Django ORM instead of SQL, it should be noted (Monod, Rouzaud, 2024).

4. System components interaction model

The Jinja2 Python template library is used to generate configuration commands. It allows you to create dynamic texts and commands using special filters, conditional statements, and variables. Templates and provided context data are combined through the Jinja2 library and the output of the prepared command is generated. The system template component uses the “Command” attribute of the “CommandTemplate” database table. It provides commands in Jinja2 format with the corresponding context data obtained in the Python program through Django ORM queries to the database. The Jinja2 template processing process is presented in the UML activity diagram of the processing process (see Figure 4). Here we see the template preparation process, which prepares commands for the configuration of switch devices. The explanations shown in the diagram indicate sample values of templates, data, and processed templates that can be applied to physical devices.

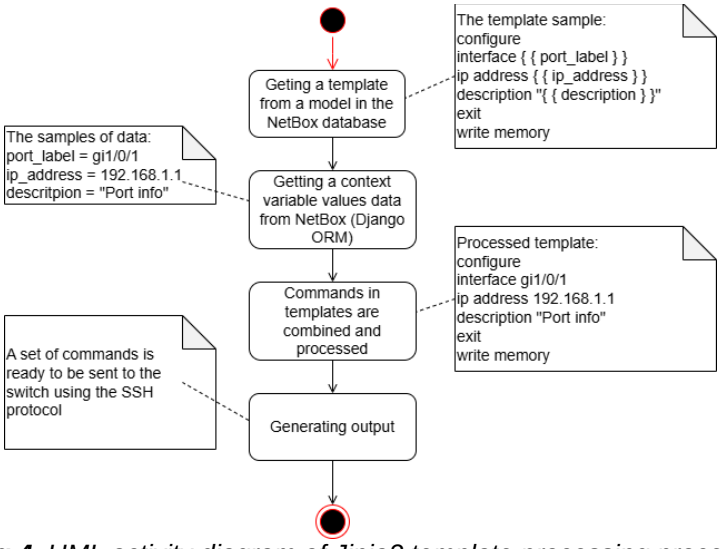

Fig 4. *UML activity diagram of Jinja2 template processing process*

The processed command sets are prepared for sending via the SSH protocol. The Paramiko and Netmiko Python libraries are used to implement the SSH protocol. Paramiko is a general SSH library, based on the functionality of which commands are executed on devices of various models, while Netmiko is a specialized and manufacturer-dependent library of specific device models, according to

2025, 29(1), 4–12. Received 10 March 2025. Accepted 24 March 2025.

which the syntax of the sent commands is determined (Osei-Wusu et al., 2025). In the Netmiko library, it is necessary to use the “DeviceTypeSelection” attribute (Figure 3) so that the syntax of the SSH command set coincides with the switch manufacturer model to which the commands are sent.

Before starting the command execution, the login data from the NetBox database is first authenticated. To access data via SSH, each switch device in the NetBox system is given the values of the physical switch for its object attributes: IP address, username and password. This data is obtained using the Python Django ORM module. In case of a successful connection, the processed command set is checked for syntax errors in the prepared template, if errors are found, the error number is returned to the user, otherwise the command set is sent to the physical switch.

The command set is split into separate lines in Python code and sent to the switch to check for errors. After that, a response from the switch is awaited and if the returned response matches the message defined in the module list (e.g., “Unrecognized command”), further command execution is stopped and an error is returned to the user. The cycle is executed until all lines of the template are checked or an error is detected. If the switch port has the described “description” value, further command execution is stopped and it is concluded that the port is already configured.

5. Discussion of system prototype testing results

When installing the NetBox server on the Windows operating system (hereinafter referred to as the OS), the caching server "Redis" was replaced with the Windows OS compatible "Memurai" (Businesswire, 2024). After preparing the NetBox server, the components DRF, Paramiko, Netmiko, Jinja2 were installed on the local computer using the Python package management tool "Pip" (Choi et al., 2024). After that, the Python directories and files required for the module were created according to the NetBox plugin development guide (NetBoxLabs, 2024). According to the database model, Python classes were described in the program files using specialized Django ORM management commands. After performing database migrations, the automatically created described models can be seen in the NetBox administrative part. After integrating Django models into the NetBox administration user interface, the process of implementing the API points required for the module being created was started. The defined HTTP methods and URL relative paths are programmatically described and linked in the view and address module files using the DRF “ViewSet” class (Christie et al., 2020). NetBox integration points can be viewed using the application programming interface documentation tool “Swagger” integrated in the NetBox platform (Ponelat, Rosenstock, 2022). The Swagger user interface allows you to work with 7 application programming interface points. To ensure security, HTTP methods are protected by special solutions: method access restrictions and usage authentication. The DRF authentication method “TokenAuthentification” is used to implement the restrictions, when users are required to specify an identification code in the request header before using each method. The identification code is assigned to each user in the NetBox platform in the “AUTH TOKEN” section of the administrative part.

Table 3. Functionality testing scenarios

Functional requirement	**Testing description**
Data retrieval from the NetBox platform	Checks whether the switch port component data is properly read and returned to the user in JSON format using the API "GET" method;
Executing command templates	Checks whether configuration API methods are properly mapped to command templates; Checks whether configuration was successfully executed for the device; Checks execution time of command templates in cases of Paramiko or Netmiko library selection.
API error handling	Using configuration API methods and command templates checks whether an intentional syntax error is detected and returned to the user in JSON format.

The purpose of system testing was to verify whether its functionality properly implements the intended scenarios. Table 3 presents the functional testing scenarios.

In order to verify the functionality of the developed system, the testing process was implemented using the following test environment components:

- A NetBox platform server running on a local computer, whose PostgreSQL database documents the initial test data;
- Python libraries for executing the SSH protocol - Paramiko and Netmiko;
- A physical (real) Dell PowerConnect 5548 model switch located in the company's network and accessible via a virtual private network;
- A GNS3 environment, which simulates a Mikrotik model switch;

2025, 29(1), 4–12. Received 10 March 2025. Accepted 24 March 2025.

- The Postman testing tool, which is used to test the module's API points.

After setting up the test environment components, we moved on to testing the first scenario – returning the read port component data to the user in JSON format. The testing process of the first scenario was performed using the Postman tool and sending a “GET” request to the “http://localhost:8000/api/plugins/switch-status-updater/VPC-testing-server20/” HTTP address. After sending the request, a response is returned to the user with the corresponding port parameter values in JSON format. Based on the provided response, it can be stated that the port parameter data was successfully read from the NetBox database and correctly returned to the user within 135 milliseconds. The remaining module requests were tested in an analogous manner. For all requests, except for "GET" and "PATCH", which specify a specific template name with different commands, the Jinja2 syntax command set was used for testing.

Testing showed that the command templates were correctly associated with the required HTTP methods and successful responses were returned to the user using the Netmiko SSH protocol library. It was found that the execution time for each request is similar (about 25 seconds) and depends on the number of commands in the template. When using a template associated with the model "Dell PowerConnect 5548" and trying to assign the Paramiko SSH protocol execution library to it, the commands are not executed correctly, because this library is not authenticated due to the other authentication type "Pubkeys" used in the switch. This problem is solved using the Netmiko library, which, when selected on the NetBox platform, requires specifying the Netmiko device type in the module's integrated user interface, on which authentication processing on the device depends.

When testing the execution of command templates in the GNS3 environment, the same initial parameters were set for the simulated switch MikroTikCRS328 SW1. After sending an HTTP POST method request, it was determined that the request was successfully executed in 1.3 seconds. The parameters of the port ether7 were manually checked in the switch management console before and after the request, the changes received corresponded to the values set in the command template. Other HTTP configurations were tested in a similar way. According to the presented results, it can be seen that when using the Paramiko SSH protocol library for the Mikrotik model device, the requests were successfully executed in an average of 0.67333 seconds. The test results with the Netmiko library also showed good results. Table 4 was compiled based on the results of the execution of command templates for both devices.

Based on the results of the second testing scenario, it can be stated that the system functionality meets the specified requirements for the command execution process. It is worth noting that although the command template execution process is faster when using the Paramiko SSH protocol library, the Netmiko library has wider compatibility with various switch device models.

Table 4. Execution time of command templates in SSH protocol libraries for different models

Device model	***SSH* protocol library**	**Average commands template execution time (s)**
„Dell Powerconnect 5548“	*Netmiko*	25.45
„Mikrotik CRS328“	*Paramiko*	0.67333
„Mikrotik CRS328“	*Netmiko*	17.34

The purpose of the third test scenario is to check how the module functionality detects syntax errors in command templates. Error detection scenarios were formed. Two test cases were assigned to each switch model. In the first and second test cases, “POST” and “PATCH” HTTP requests were sent using the Netmiko SSH protocol library. In the second test case when the error “port_labiel” was made in the variable name instead of “port_label”, a message about the missing part of the command was returned to the user. This happened because after changing the variable name, the module functionality could not find a documented match for the port parameter from the NetBox database, and in this case, no value was inserted instead of the variable and one parameter was missing in the command line. Based on the API responses received in the first and second cases, it can be stated that the JSON response was properly formed and errors were detected purposefully during the command execution process on the switch. Error detection testing showed that the system functionality detects expected errors and returns them to the user in JSON format.

The tools selected during the testing process were properly adapted for testing the system's functionality. The initial data required for the testing process was successfully stored on the NetBox platform. Using the Postman functionality of the tool, the module's API request command execution process was assessed for compliance with the functional requirements of the project, and the functionality of the GNS3 platform provided the opportunity to simulate a virtual Mikrotik model switch and use it for testing. Based on the testing results, it can be stated that the selected SSH protocol library Netmiko was more compatible with switches of different models and required less command syntax adaptation.

2025, 29(1), 4–12. Received 10 March 2025. Accepted 24 March 2025.

Conclusions

1. Analysis of the NetBox platform user interface showed that during the execution of the system configuration, this platform can document descriptions, Internet protocols, MAC addresses and other switch port parameters in the PostgreSQL database, process them using integrated Python programs, linking Django objects with a relational connection, applying REST API methods;
2. A system model was designed, with an integrated API component and database, an interaction model of system components was prepared, using the SSH protocol Paramiko or Netmiko libraries, the NetBox platform REST API integration points were supplemented with 7 methods used for configuring switch ports, protecting them with Token authentication;
3. System prototype testing showed that the module functionality meets the intended requirements of the command execution process. According to the results of scenario testing, it was determined that the port parameter data was successfully read from the NetBox database and properly returned to the user within an average of 135 milliseconds. It was found that the configuration execution time for switches depends on the number of command lines in the command templates. It was determined that the selected SSH protocol library Netmiko, despite the longer command execution time, was better compatible with switches of different models and required less command syntax adaptation.

References

1. bin Uzayr, S. (2022). *Mastering Visual Studio Code: A Beginner's Guide*. CRC Press.
2. Businesswire. (2024). *Redis on Windows? Redis Partners with Memurai for Windows Compatibility*. Prieiga per internetą: https://www.businesswire.com/news/home/20241002905636/en/Redis-on-Windows-Redis-Partners-with-Memurai-for-Windows-Compatibility. Žiūrėta 2024-12-15.
3. Cao, Y., Chen, Z., Zhang, X., Li, Y., Chen, L., & Wang, L. (2024). Diagnosis of package installation incompatibility via knowledge base. *Science of Computer Programming*, *235*, 103098.
4. Choi, B. (2024a). Installing NetBox (IPAM/DCIM) with Python. In *Introduction to Python Network Automation Volume II: Stepping up: Beyond the Essentials for Success* (pp. 715-767). Berkeley, CA: Apress.
5. Choi, B. (2024b). Python Network Automation Labs: SSH in Action, paramiko and netmiko Labs. In *Introduction to Python Network Automation Volume II: Stepping up: Beyond the Essentials for Success* (pp. 121-227). Berkeley, CA: Apress.
6. Chrisna, I. D. A. I. W., Kusumo, D. S., ir Riskiana, R. R. (2024). *LOW CODE INTEGRATION TESTING IN OUTSYSTEMS PERSONAL ENVIRONMENT*. *Jurnal Teknik Informatika (Jutif)*, *5*(2), 551-560. Prieiga per internetą: https://www.jutif.if.unsoed.ac.id/index.php/jurnal/article/download/1673/476. Žiūrėta 2024-12-15.
7. Christie, M., Marru, S., Abeysinghe, E., Upeksha, D., Pamidighantam, S., Paul Adithela, S., ...& Pierce, M. (2020). An extensible django-based web portal for apache airavata. In *Practice and Experience in Advanced Research Computing 2020: Catch the Wave* (pp. 160-167).
8. Dhadil, S., Srivastava, A., Shinde, V., Walhekar, V., Patil, A., Ganeshpurkar, A.,. & Kulkarni, R. (2024). Django Unleashed: A Deep Dive into the Features and Advantages of the Django Framework. *RGUHS Journal of Pharmaceutical Sciences*, *14*(3).
9. Django. (2024). *Making queries*. Prieiga per internetą: https://docs.djangoproject.com/en/5.1/topics/db/queries/. Žiūrėta 2024-12-15.
10. Dzemydienė, D., Turskienė, S., & Šileikienė, I. (2023). Development of ICT infrastructure management services for optimization of administration of educational institution activities by using ITIL-v4. *Baltic journal of modern computing.*, *11*(4), 558-579.
11. Gupta, T. (2024, December). Kubernetes-Driven Network Security for Distributed ACL Management. In *2024 8th Cyber Security in Networking Conference (CSNet)* (pp. 236-242). IEEE.
12. Haibeh, L. A., Yagoub, M. C., & Jarray, A. (2022). A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches. *IEEE Access*, *10*, 27591-27610.
13. Irtaza, W. (2021). *IT Infrastructure Automation Using Ansible: Guidelines to Automate the Network, Windows, Linux, and Cloud Administration (English Edition)*. BPB Publications.
14. Jordon, W. (2019). Python django web development: The ultimate django web framework guide for Beginners. *Kindle Edition*.
15. Kore, P. P., Lohar, M. J., Surve, M. T., & Jadhav, S. (2022). API Testing Using Postman Tool. *International Journal for Research in Applied Science and Engineering Technology*, *10*(12), 841-43.

2025, 29(1), 4–12. Received 10 March 2025. Accepted 24 March 2025.

16. Lin, E., Koishybayev, I., Dunlap, T., Enck, W., & Kapravelos, A. (2024, February). UntrustIDE: Exploiting Weaknesses in VS Code Extensions. In *Proceedings of the ISOC Network and Distributed Systems Symposium (NDSS). Internet Society*.

17. Lucifora, R. (2024). *Improving Observability in Large Enterprise Networks with NetBox And SuzieQ* (Doctoral dissertation, Politecnico di Torino).

18. Monod, O., & Rouzaud, D. (2024, May). Towards OGC API-Features Centric GIS Applications Controlled by Object Relational Mapping. In *International Symposium on Web and Wireless Geographical Information Systems* (pp. 105-113). Cham: Springer Nature Switzerland.

19. Mulyana, E., & Fakih, G. (2022). Network Automation with a Single Source of Truth in a Heterogeneous Environment. *International Journal on Electrical Engineering & Informatics*, *14*(1).

20. Nketsiah, R. N., Millham, R. C., Agbehadji, I. E., Freeman, E., & Epizitone, A. (2023, October). Optimising a Formulated Cost Model to Minimise Labour Cost of Computer Networking Infrastructure: A Systematic Review. In *International Conference on Advanced Research in Technologies, Information, Innovation and Sustainability* (pp. 427-442). Cham: Springer Nature Switzerland..

21. Osei-Wusu, F., Asiedu, W., Frimpong, K. A., Yeboah, D., & Muntaka, S. A. (2025). A Scheme for Network Programmability and Backup Automation Using Python Netmiko Library on Cisco; the Case Study of the Komfo Anokye Teaching Hospital Local Area Network.

22. Ponelat, J., & Rosenstock, L. (2022). *Designing APIs with Swagger and OpenAPI*. Simon and Schuster.

23. Priyadarsini, M., & Bera, P. (2021). Software defined networking architecture, traffic management, security, and placement: A survey. Computer Networks, 192, 108047.

24. Šoška, M. (2021). System for automatic discovery of network devices and their topology.

25. Uramova, J., Segec, P., & Moravcík, M. (2024, October). Contribution to Safer Internet for Children at School. In *2024 International Conference on Emerging eLearning Technologies and Applications (ICETA)* (pp. 1-7). IEEE.

26. Zhu, X. (2021). Self-organized network management and computing of intelligent solutions to information security. *Journal of Organizational and End User Computing (JOEUC)*, *33*(6), 1–16.