Tampere University of Applied Sciences

FastAPI as a Backend Frame-work

Xiaomin Wu

BACHELOR'S THESIS
May 2024

Bachelor's Degree Program in Software Engineering

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Bachelor’s Degree Program in Software Engineering

AUTHOR: Xiaomin Wu
Title of Thesis: FastApi as Backend Framework

Bachelor's thesis 25 pages.
May 2024

This thesis studies the integration of FastAPI, a modern Python web framework, into the Actiweb project, a Building Automation System (BAS), to enhance operational efficiency and responsiveness. Actiweb, developed by Bithouse Oy, specializes in automation solutions for building management systems. The transition from Actiweb 8 to Actiweb 9 involves replacing the PHP backend with FastAPI to meet performance demands.

The technical background section mainly describes the requirements, design considerations, and implementation details. FastAPI's architecture, features, and advantages over other frameworks, notably Node.js, are discussed, highlighting its asynchronous capabilities, memory management, and UI automation stability.

Implementation details cover tools, techniques, and features such as rendering frontend logic, managing endpoints, and implementing authorization through token-based authentication. Evaluation criteria ensure seamless rendering of frontend logic and reliable authorization mechanisms.

Challenges and solutions focus on addressing the high asynchronous requirements of BAS systems, leading to the adoption of FastAPI. Future features include differentiated user security levels and integration into the Actiweb project.

In summary, the use of FastAPI in Actiweb enabled the evolution of the project and helped modernize the BAS system. This paper also highlights the importance of choosing the right tools and techniques in software engineering projects.

Key words: fastapi, python, backend, bas, web-development

CONTENTS

ABBREVIATIONS AND TERMS

PHP	PHP: Hypertext Preprocessor
IDE	Integrated Development Environment
BAS	Building Automation System
BMS	Building Management System
BAC	Building Automation Controller
HVAC	Heating, Ventilation, Air Conditioning
API	Application Programming Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheet
GDPR	General Data Protection Regulation
HIPAA	Health Insurance Portability and Accountability Act

1 INTRODUCTION

This report pertains to a collaborative training initiative undertaken with Bithouse Oy, an automation software firm in Tampere. Bithouse Oy specializes in the development of automation detection and control solutions. The company comprises a workforce of 15 individuals, encompassing various support functions such as marketing and logistics, alongside a predominant focus on hardware and software development.

This report focuses on a part of the transition from Actiweb 8 to Actiweb 9, representing an evolution in the automation platform's capabilities. Actiweb 9 introduces a browser-based user interface, replacing the previous version's backend, which utilized the PHP programming language without any framework. Responding to customer demands for enhanced performance, Actiweb 9 incorporates backend upgrades and integration of the FastAPI framework, thus augmenting operational efficiency and responsiveness.

This report aims to explore the utilization of FastAPI, a modern Python web framework, as a potent tool for automating backend processes within the realm of web application development. FastAPI's unique features, including asynchronous support and intuitive API creation, make it an ideal candidate for optimizing and streamlining various backend automation tasks.

2 Background

This chapter mainly describes the general situation of the company (Bithouse Oy) and project (Actiweb). And leads to the topic of the paper, FastApi as Backend Framework.

2.1 Company

According to Bithouse website (Bithouse n.d.) Bithouse Oy is a Tampere-based technology company that specializes in providing innovative solutions to various industries, with a focus on building automation systems. The company sells self-developed automation platforms or controller modules, including but not limited to Fire damper control and monitoring system, alarm center and cloud monitoring tool. At the same time, the company provides customized services that could be designed and implemented including small and large electronics, programming projects, bus communication solutions and extensive cloud service applications to meet the unique needs of customers.

The company comprises a workforce of 15 individuals, encompassing various support functions such as marketing and logistics, alongside a predominant focus on hardware and software development.

2.2 Project

Actiweb is a platform for a Building Automation System (BAS) (Bithouse n.d.) consisting of a Building Automation Controller (BAC) and a Building Management System (BMS). Its primary function is to monitor and control various building systems and equipment to optimize energy usage, ensure occupant comfort, and enhance overall operational efficiency.

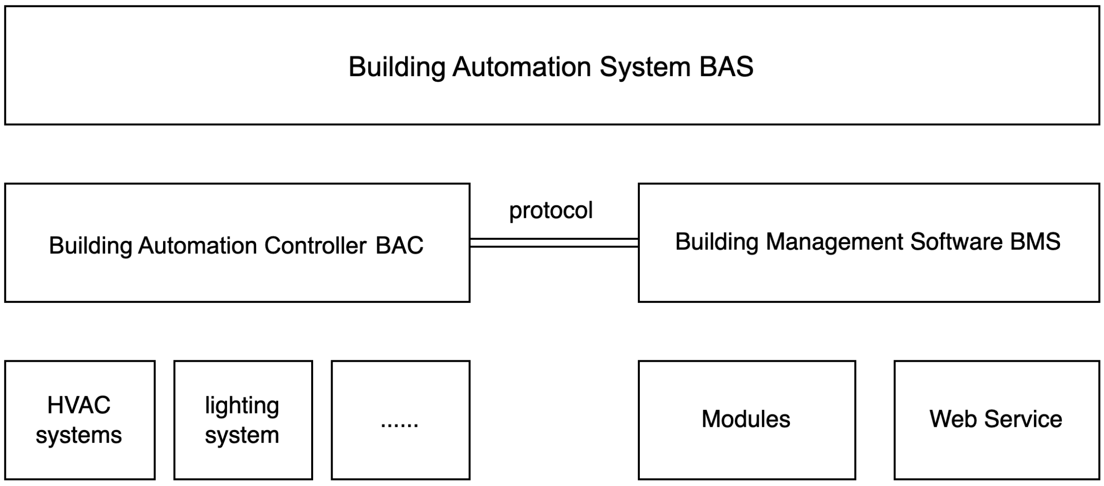

Building Automation System BAS
Building Automation Controller BAC
protocol
Building Management Software BMS
HVAC systems
lighting system
......
Modules
Web Service

FIGURE1: BMS software application diagram

BACs are physical hardware devices specifically designed for building automation tasks. These devices often include dedicated microcontrollers or processors, along with input/output interfaces for connecting to sensors, actuators, and other building systems.

Building Management Software (BMS) typically runs on standard computing hardware such as servers or workstations. This software provides a user-friendly interface for building operators to manage and control building systems, schedule operations, visualize and analyze data, and generate reports.
While BMS software can handle higher-level functions such as scheduling and data analysis, it typically relies on the BAC to execute control logic and communicate with individual building systems.

As mentioned above, the Actiweb project as a BAS contains two parts, BAC and BMS. The BMS uses a specific protocol to control the BAC to implement specific operations, including environmental monitoring and data analysis, and these functions can be directly accessed by the browser. The main programming language used in the project is Lua, which is a lightweight, efficient, and versatile programming language designed for embedded systems, scripting, and general-purpose programming. (Lua n.d.)

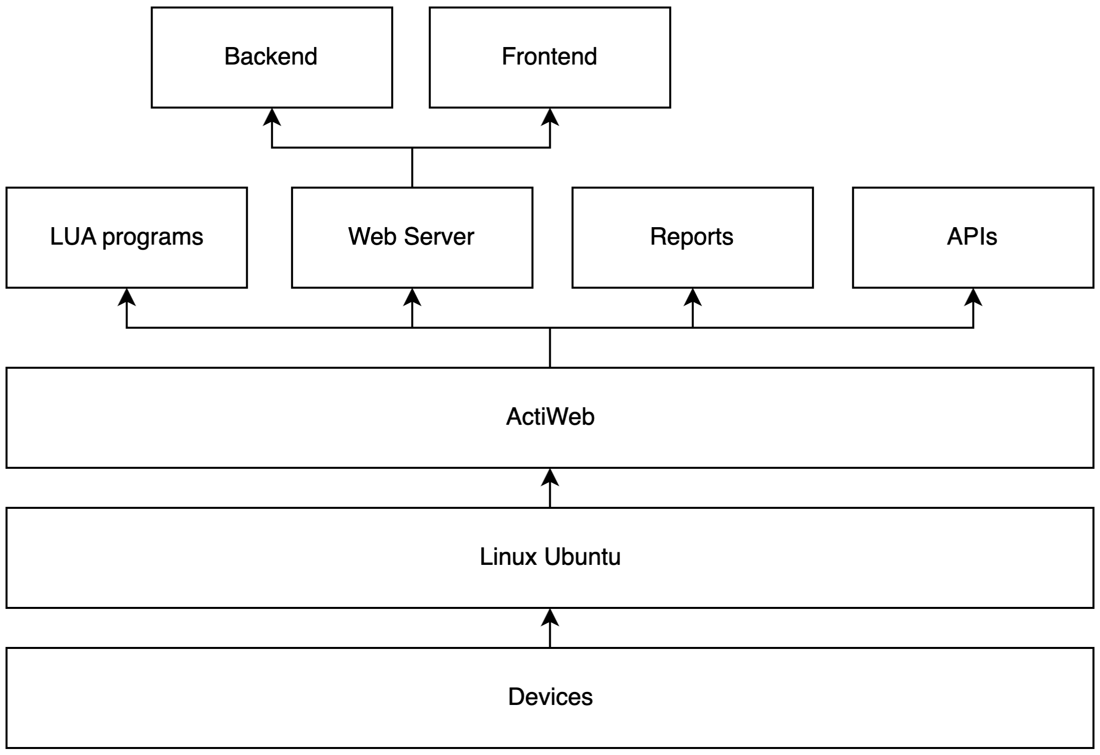


FIGURE2: Structure of ActiWeb (Bithouse n.d.)

The picture above shows the structure of the Actiweb project. The Actiweb platform essentially includes basic functions such as automation programs (LUA), web server and various communication APIs, running on Linux Ubuntu devices. The content of this report focuses on the web server, especially the backend part.

2.3 Update the backend with FastAPI framework

This thesis aims to explore the utilization of FastAPI, a modern Python web framework, as a formidable tool for automating backend processes within web application development. (FastAPI n.d.)

The upgrade project of Actiweb is split into small parts, which includes the backend update, the main topic of this report. Actiweb 9 uses FastAPI and Python program language to replace Actiweb 8 which uses PHP without any framework.

Python is commonly used in backend development due to its simplicity, versatility, and a robust ecosystem of libraries and frameworks. (Python n.d.) Its clean and readable syntax makes it easy to develop and maintain complex applications, and its extensive standard library and third-party packages offer solutions for various tasks, including web development, data processing, and automation.

The report will continue to discuss this topic below, regarding the specific project implementation of FastAPI as a backend framework.

3 Technical Background

3.1 Requirements

This chapter describes the requirements for the backend, divided into functional requirements and non-functional requirements. Functional requirements specify system-specific functions or features, such as user authentication or maintenance management. Non-functional requirements define how the system should perform, including aspects of performance, scalability, security, reliability, compliance and user experience. These requirements lay the foundation for developing a robust and reliable building management software web backend that meets the needs of property managers, tenants, and other stakeholders.

3.1.1 Functional Requirements

As the user interface of an environmental monitoring system, it needs to allow users to monitor environmental parameters in real time, set alarm conditions, view historical data, and perform necessary operations.

User authentication is an important requirement to ensure information security. Users should be able to register, log in and out securely, with different access levels as different user roles (e.g. administrator, normal user).

Real-time asynchrony requirements are also important functional requirements for this web page, which provides real-time updated environmental parameter data, including temperature, humidity, air quality, etc., so that users can understand the current environmental status at any time.

Additionally, the web page needs to meet functional requirements: alerts and notifications, history and reporting. It allows users to set alert conditions and send alert notifications when conditions are triggered so that necessary actions can be taken in a timely manner. It also needs to provide historical data recording functionality, allowing users to view past environmental parameter data and be able to generate reports for analysis or sharing.

3.1.2 Non-functional Requirements

Non-functional requirements mainly improve the user experience, based on meeting functional requirements and related regulations and standards, such as GDPR or HIPAA.

The user interface should be designed to be concise and clear, easy to understand and operate, so that users can quickly find the information and functions they need. Data is graphed through charts, graphs, or other visualization methods to display the historical trends of environmental parameters to help users analyze data and make decisions. The interface should be intuitive and easy to use, with clear navigation and informative error messages.

ActiWeb supports multiple devices and screen sizes to accommodate users accessing the system from a variety of devices.

Picture1: Actiweb software is flexible using on different devices. (Bithouse n.d.)

3.2 Design Consideration

3.2.1 Architecture

The architectural structure diagram of the Actiweb project can be summarized as the following figure. The user interface contains frontend and backend, where the frontend interacts with the user and the backend interacts with the SLC Engine that processes data and operations.

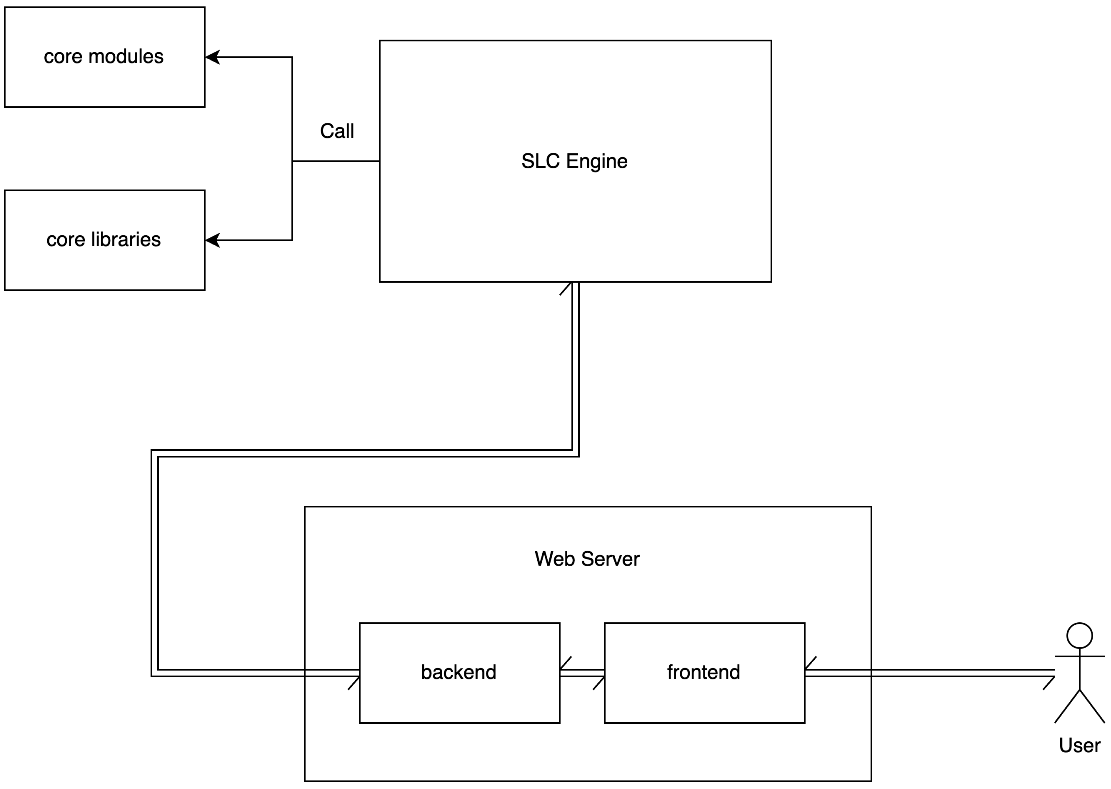


FIGURE3: Architecture of Actiweb project.

The topic of the thesis is mainly related to backend, which is closely related to the SLC engine (where data processing is completed) and the UI of frontend. Therefore, the architect who designed and tutored this project selected the FastApi framework to meet the needs of efficient multi-threaded asynchronous processing, more stable memory management and UI automation solutions.

3.2.2 FastApi

Fastapi is an intuitive backend framework for building APIs with Python 3.8+ based on standard Python type hints that works with OpenAPI automated documentation. (FastAPI n.d.)

By leveraging OpenAPI specifications, FastAPI streamlines the process of documenting APIs, ensuring consistency and accuracy across endpoints. This automated documentation not only aids developers in understanding and utilizing the API efficiently but also enhances collaboration and communication within development teams. Additionally, FastAPI's intuitive design, powered by Python's type hinting system and automatic data validation, further accelerates development by minimizing boilerplate code and reducing the likelihood of errors. The framework's asynchronous capabilities enable high-performance, scalable backend applications, making it an appealing choice for modern web development projects where speed and efficiency are paramount.

Additionally, Python's popularity among developers and its large community ensures abundant resources, documentation, and support, making it an attractive choice for backend development projects of all sizes and complexities.

3.2.3 Compared to other frameworks

This chapter mainly uses Node.js as an example to explain the advantages of FastApi compared to other backend frameworks.

According to a blog post by NextGenerationAutomation Node.js runs on a single-threaded event loop (NextGenerationAutomation n.d.), which can cause performance bottlenecks, especially when dealing with heavy data processing tasks. FastAPI is built on Starlette and Pydantic, making full use of Python's asynchronous capabilities to achieve efficient concurrency and parallelism. This means that FastAPI can handle CPU-intensive tasks more efficiently, leverage asynchronous programming paradigms, and even scale across multiple CPU cores when needed.

Node.js applications can experience memory leaks and high memory consumption, especially when processing large data sets. This can lead to instability and crashes, especially in long-running data processing tasks where memory allocation cannot be managed efficiently. Python's memory management is generally more stable and predictable compared to Node.js. FastAPI's reliance on Python memory management means it can handle large data sets and long-running processes more efficiently without worrying about memory-related issues.

Node.js lacks cross-browser testing, strong element positioning strategies, and standardized solutions for handling complex UI interactions, which can make UI automation more challenging and error-prone. While FastAPI focuses primarily on backend development, Python has several mature UI automation libraries, such as Selenium WebDriver and PyAutoGUI. These libraries provide powerful solutions for automating web and desktop UIs, with extensive functionality for element positioning, interaction, and cross-browser testing. Compared to Node.js, which may lack mature UI automation solutions, the combination of FastAPI and Python libraries provides a more stable and reliable platform for automating user interfaces.

Overall, FastAPI has several advantages over Node.js, especially in terms of concurrency, memory management, data processing capabilities, and UI automation stability.

4 Implementation

4.1 Tools and techniques used in practice

The project has a comprehensive suite of tools to streamline development processes and optimize project management.

The primary integrated development environment (IDE) is VSCode, providing a robust platform for coding and project organization. Additionally, this project leverages VirtualBox machines to create versatile development environments, enhancing flexibility and resource allocation. For asynchronous programming tasks, the project utilizes Uvicorn, which is commonly used as an ASGI (Asynchronous Server Gateway Interface) server to serve web applications developed with frameworks like FastAPI or Starlette, providing high-performance asynchronous handling for modern web development projects.

The development stack includes Python, HTML, CSS, and JavaScript, enabling us to create dynamic and interactive web applications. Among them, HTML, CSS, and JavaScript are used for frontend development and are inherited from actiweb8. They only need to be fine-tuned to match the new backend. The backend will be implemented in Python using the FastAPI framework, providing server-side logic and API endpoints to support the functionality of the frontend. The development stack would also include a database solution but it was implemented by a different developer and as this project is still in development the database language is not decided yet.

Moreover, OpenProject and Clockify are used for tracking working time and task processes, the project team relies on them to monitor and analyze resource allocation across various project tasks.

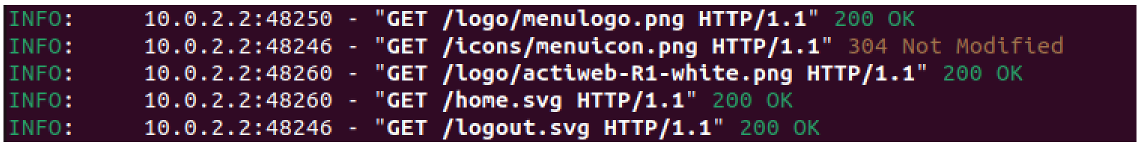
Together, these tools enable the project to streamline workflows, enhance collaboration, and ensure efficient resource utilization throughout the project lifecycle.

4.2 Features and Functions

4.2.1 Render

This project inherits the front-end code of the previous version (including HTML CSS and JavaScript) and mounts them to the backend directory. In this way, when launching the web page, the front-end code will automatically complete rendering.

```
from fastapi import FastAPI, Depends
from fastapi.staticfiles import StaticFiles

app = FastAPI()

app.mount("/static", StaticFiles(directory="/var/www/html"),name="index.html")
app.mount("/images",StaticFiles(directory="/var/www/html/images"),name="")
app.mount("/pages",StaticFiles(directory="/var/www/html/pages"),name="")
```
CODE 1: Code snippet used to mount frontend component

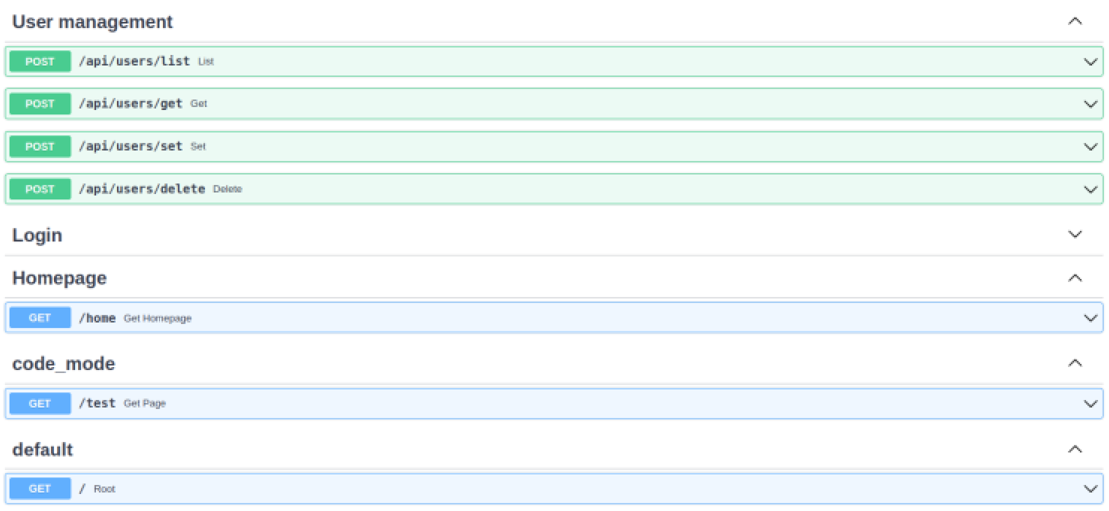

```
INFO:     10.0.2.2:48250 - "GET /logo/menulogo.png HTTP/1.1" 200 OK
INFO:     10.0.2.2:48246 - "GET /icons/menuicon.png HTTP/1.1" 304 Not Modified
INFO:     10.0.2.2:48260 - "GET /logo/actiweb-R1-white.png HTTP/1.1" 200 OK
INFO:     10.0.2.2:48260 - "GET /home.svg HTTP/1.1" 200 OK
INFO:     10.0.2.2:48246 - "GET /logout.svg HTTP/1.1" 200 OK
```
Picture 2: screenshot of an example for successfully rendering the frontend component

4.2.2 Endpoints

FastApi provides a clear and concise endpoints management method. Reference and declare the required endpoints in main.py, which has its function in its own file.

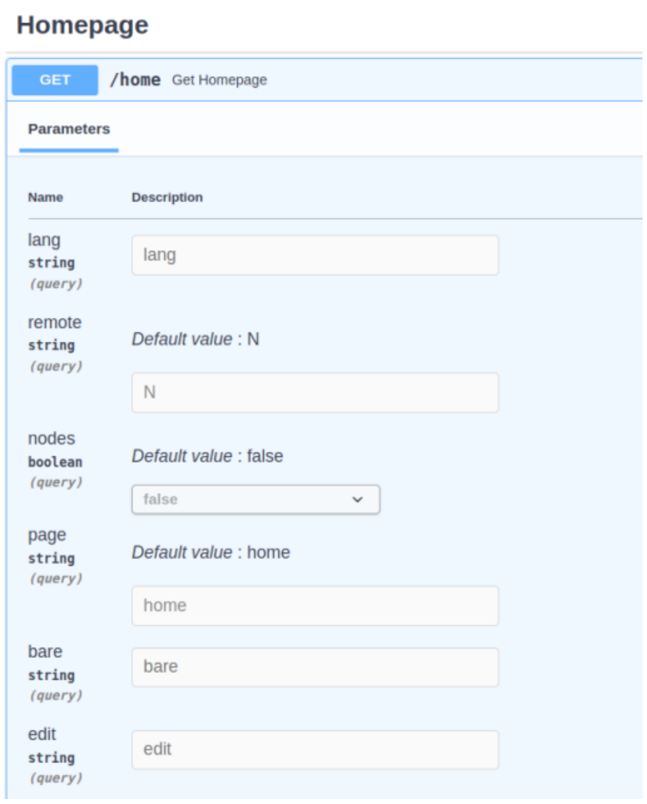
```
from api.routers import cmd, auth, server, login, home

app.include_router(cmd.router, prefix = ”/api”)
app.include_router(auth.router, prefix = ”/api”)
app.include_router(server.router, prefix = ”/api”)

app.include_router(login.router)
app.include_router(home.router)
```
CODE 2: Code snippet used to define endpoints

FastApi provides a clear and concise way to manage endpoints. Reference and declare the required endpoint in main.py, which has its functionality in its own file. After declaring the endpoint, FastApi provides automatic interactive API documentation to facilitate testing and monitoring api functions.



Picture 3: Screenshot of FastApi automatic interactive API documentation

In the automatic interactive API document, the developer can view the parameters and responses of the API. Developers can also test to see if they can get the expected response.



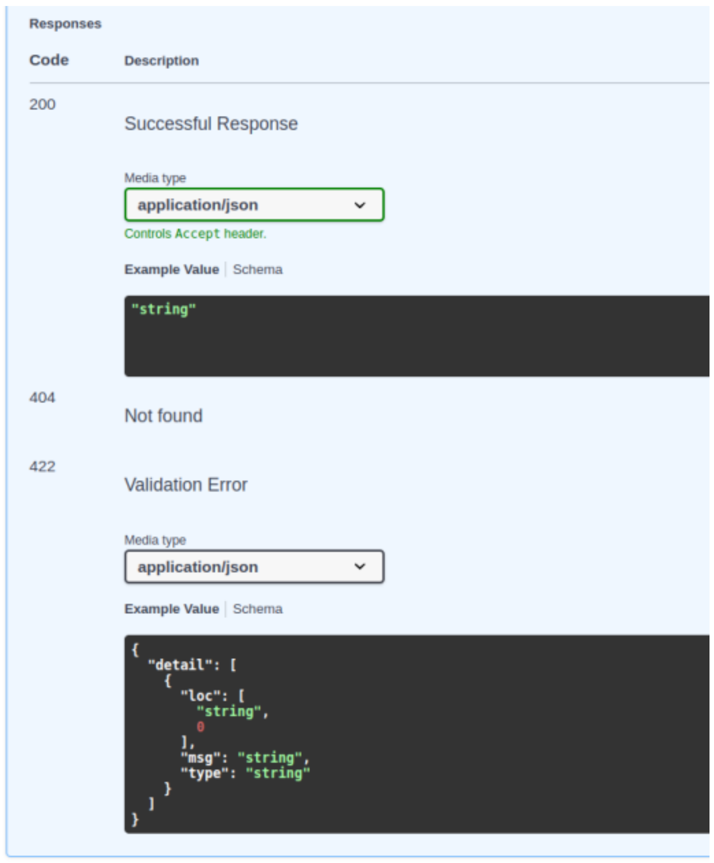
Homepage

GET /home Get Homepage

Parameters

Name	Description
lang string *(query)*	lang
remote string *(query)*	*Default value* : N N
nodes boolean *(query)*	*Default value* : false false
page string *(query)*	*Default value* : home home
bare string *(query)*	bare
edit string *(query)*	edit

Picture 4: Screenshot of parameters for this endpoint

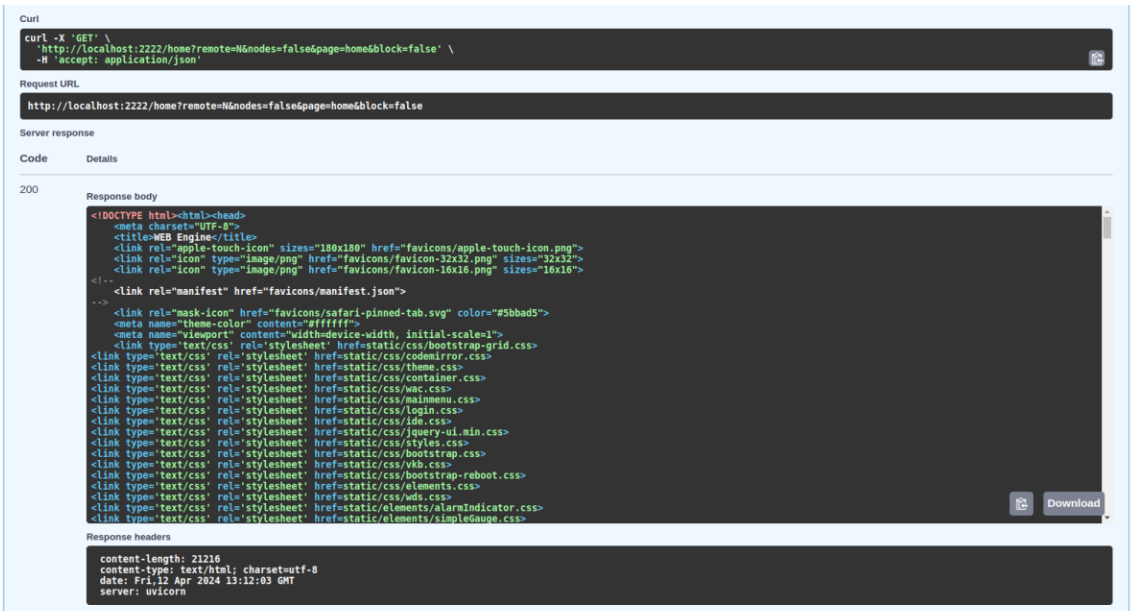


Responses

Code	Description
200	Successful Response Media type: application/json Controls Accept header. Example Value | Schema "string"
404	Not found
422	Validation Error Media type: application/json Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

Picture 5: Screenshot of responses for this endpoint


Curl

```
curl -X 'GET' \
  'http://localhost:2222/home?remote=N&nodes=false&page=home&block=false' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:2222/home?remote=N&nodes=false&page=home&block=false
```

Server response

Code Details

200

Response body

```
<!DOCTYPE html><html><head>
    <meta charset="UTF-8">
    <title>WEB Engine</title>
    <link rel="apple-touch-icon" sizes="180x180" href="favicons/apple-touch-icon.png">
    <link rel="icon" type="image/png" href="favicons/favicon-32x32.png" sizes="32x32">
    <link rel="icon" type="image/png" href="favicons/favicon-16x16.png" sizes="16x16">
<!--
    <link rel="manifest" href="favicons/manifest.json">
-->
    <link rel="mask-icon" href="favicons/safari-pinned-tab.svg" color="#5bbad5">
    <meta name="theme-color" content="#ffffff">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link type='text/css' rel='stylesheet' href=static/css/bootstrap-grid.css>
<link type='text/css' rel='stylesheet' href=static/css/codemirror.css>
<link type='text/css' rel='stylesheet' href=static/css/theme.css>
<link type='text/css' rel='stylesheet' href=static/css/container.css>
<link type='text/css' rel='stylesheet' href=static/css/wac.css>
<link type='text/css' rel='stylesheet' href=static/css/mainmenu.css>
<link type='text/css' rel='stylesheet' href=static/css/login.css>
<link type='text/css' rel='stylesheet' href=static/css/ide.css>
<link type='text/css' rel='stylesheet' href=static/css/jquery-ui.min.css>
<link type='text/css' rel='stylesheet' href=static/css/styles.css>
<link type='text/css' rel='stylesheet' href=static/css/bootstrap.css>
<link type='text/css' rel='stylesheet' href=static/css/vkb.css>
<link type='text/css' rel='stylesheet' href=static/css/bootstrap-reboot.css>
<link type='text/css' rel='stylesheet' href=static/css/elements.css>
<link type='text/css' rel='stylesheet' href=static/css/wds.css>
<link type='text/css' rel='stylesheet' href=static/elements/alarmIndicator.css>
<link type='text/css' rel='stylesheet' href=static/elements/simpleGauge.css>
```

Response headers

```
content-length: 21216
content-type: text/html; charset=utf-8
date: Fri,12 Apr 2024 13:12:03 GMT
server: uvicorn
```

Picture 6: screenshot of Api test via FastApi automatic interactive API documentation

4.2.3 Authorization

After the user enters the password, a token is generated and stored in the browser's cookie. After that, any API operation will first check whether the token exists and whether the token meets the permission requirements. If there is no token, the user can only see the login interface. This function that users with different permissions can access information with different privacy levels is also implemented through different tokens.

```python
from fastapi import APIRouter, Depends, HTTPException, Response
from ..auth import common as auth
from ..types.auth import Token

router = APIRouter(
    prefix="/auth",
    tags=["Authentication"],
    dependencies=[],
    responses={404: {"description": "Not found"}},
)

    #--------------------AUTHENTICATION----------------------

@router.post("/pw", response_model=Token)
```

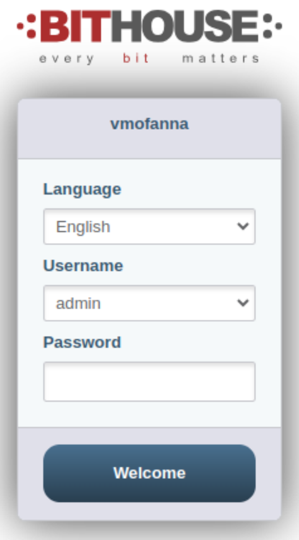
```
async def token_login(response: Response, token_data: dict =
Depends(auth.pw_login), redirect: bool = True):
    response.set_cookie(key="token_data", value=token_data)

    if redirect:
        response.headers["Location"] = "/home"
        response.status_code = 302                    # indicating a redirect
        return response

    return token_data
```
CODE 3: Code snippet used to generate token

```
def get_current_user(request: Request):
    if 'token_data' in request.cookies:
        token_str = request.cookies.get("token_data")
        token_str = token_str.replace("'","\"")
        token = json.loads(token_str)
```
CODE 4: Code snippet used to check the token

5 Eveluation

5.1 Evaluation criteria

Since this project is an upgrade project based on the eighth edition and inherits the frontend code of the previous version, the most basic requirement for the backend is that the web page can render the frontend logic as before.



Picture 7: screenshot of the login form

Secondly, an important point in this version of the upgrade is authorization, which distinguishes users with different permissions. This is accomplished by using to-kens, discussed in detail in the previous chapter.

6 Challenge and solution

As introduced in the previous content, this backend project serves a BAS (building automation system), so the system has high asynchronous requirements for the backend. The original PHP program is difficult to meet that requirement, so the system needs to be upgraded. During the upgrade process, introducing the FastApi framework with python as the programming language, which is the problem studied and solved in this paper.

7 Future Features

In future functions, user security levels will be differentiated, and different users can access different levels of data and functions. This function will be implemented by analyzing different tokens to obtain different user information.

In addition, after the backend is completed, other colleagues will integrate the web service into the Actiweb project to achieve complete system functions.

8 DISCUSSION

This thesis explored the adoption of FastAPI as a backend framework for the Actiweb project, a Building Automation System (BAS) focusing on enhancing operational efficiency and responsiveness.

The transition from Actiweb 8 to Actiweb 9 marks an important evolution in the platform's capabilities, particularly with the integration of FastAPI to replace the previous PHP backend. This upgrade addresses customer demands for improved performance and introduces modern features such as asynchronous support and intuitive API creation.

The selection of FastAPI as the framework was made based on a few important reasons; It benefits from the simplicity and versatility of Python, which makes FastAPI a robust framework for developing backend solutions. Its asynchronous capabilities enable real-time data processing, while its automatic documentation feature makes API development easier.

Additionally, the evaluation criteria demonstrated the successful implementation of the backend update, ensuring seamless rendering of frontend logic and robust authorization mechanisms through token-based authentication. This achievement underscores the effectiveness of FastAPI in meeting the project's requirements and delivering a reliable backend infrastructure for the Actiweb platform.

Future features, such as differentiated user security levels and seamless integration into the Actiweb project make future updates and support easier to manage in the future.

In conclusion, using FastAPI as a backend framework for the Actiweb project is an important step towards modernizing and optimizing the ActiWeb system. This project contributes to advancing web development practices and highlights the importance of selecting the right tools and technologies to be efficient in software engineering projects.

REFERENCES

Bithouse Oy homepage
Read on 29.4.2024
https://bithouse.fi/

Bithouse Oy ActiWeb product page
Read on 29.4.2024
https://en.bithouse.fi/actiweb-automation-platform/

FastAPI Documentation
Read on 29.4.2024
https://fastapi.tiangolo.com/

Lua documentation
Read on 29.4.2024
https://www.lua.org/about.html

Python Documentation
Read on 29.4.2024
https://www.python.org/about/

Understand Node JS Single Thread Event Loop Work Flow
NextGenerationAutomation
Read on 29.4.2024
https://www.nextgenerationautomation.com/post/understand-node-js-single-thread-event-loop-work-flow#:~:text=Event%20Loop%20uses%20Sin-gle%20Thread,for%20incoming%20requests%20for%20indefinitely.