## A REPORT

## ON

**PYTHON DEVELOPER, SEMICON DOMAIN**

***Submitted by,***

**BHAGYASHREE C. PATIL - 20211CSE0372**

### *Under the guidance of,*

**Dr. VIJAYA KUMAR A V**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**At**



**PRESIDENCY UNIVERSITY**

**BENGALURU**

**MAY 2025**

**PRESIDENCY UNIVERSITY**

**PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**

This is to certify that the Internship/Project report **“PYTHON DEVELOPER, SEMICON DOMAIN”** being submitted by “BHAGYASHREE C. PATIL” bearing roll number “20211CSE0372” in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a bonafide work carried out under my supervision.

|  |  |
| --- | --- |
| **Dr. VIJAYA KUMAR A V**  Professor  PSCS / PSIS  Presidency University | **Dr. ASIF MOHAMMED**  HoD  PSCS  Presidency University |

|  |  |
| --- | --- |
| **Dr. MYDHILI NAIR**  Associate Dean  PSCS  Presidency University | **Dr. SAMEERUDDIN KHAN**  Pro-Vice Chancellor - Engineering  Dean –PSCS / PSIS  Presidency University |

**PRESIDENCY UNIVERSITY**

**PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**DECLARATION**

I hereby declare that the work, which is being presented in the report entitled “**PYTHON DEVELOPER, SEMICON DOMAIN”** in partial fulfillment for the award of Degree of **Bachelor of Technology** in **Computer Science and Engineering**, is a record of my own investigations carried under the guidance of **Dr. VIJAYA KUMAR A V,** **Presidency School of Computer Science and Engineering, Presidency University, Bengaluru.**

I have not submitted the matter presented in this report anywhere for the award of any other Degree.

|  |  |
| --- | --- |
|  | **BHAGYASHREE C. PATIL**  **20211CSE0372** |

**INTERNSHIP COMPLETION CERTIFICATE**

****

**ABSTRACT**

This is a detailed overview of the technical work and activity performed during my internship at L&T Technology Services (LTTS). Throughout the internship, I was able to work as a backend developer on Python, which helped in API development as part of a comprehensive full-stack in-house solution. This project, spearheaded under collaboration with LTTS and the R&D Datacenter Management team, was focused on safely polling real-time data from lab environments, with the ultimate objective of guaranteeing that polling mechanisms did not interrupt current lab executions or overload system resources.

Working on this project gave me the opportunity to get hands-on experience in backend development, where I researched and implemented best practices in secure API design, scalable architecture, and readiness for real-world deployment. I became experienced in working with FastAPI, a modern Python web framework, to design asynchronous, high-performance RESTful APIs. The project emphasized data protection and integrity, which led me to dive into cryptographic techniques, including AES encryption (CBC mode) for secure password transmission and bcrypt hashing for safe storage.

Besides API development, I gained hands-on experience in PostgreSQL, learning to execute database operations, design schemas properly, and map data models to the backend services with clean and efficient code. I also used object-oriented programming (OOP) concepts throughout the project to make code more maintainable and allow modular development.

One of the most challenging parts of the internship was adherence to PEP8 coding standards, environment variable management, and structured logging setup to facilitate debuggability as well as production-grade observability. This helped in establishing a clean, consistent, and scalable codebase with adherence to industry’s best practices.

This internship was an important steppingstone between theoretical education and real-world application. It honed my technical skills, especially in secure backend development and scalable system design. I gained experience working in a structured development environment, handling feedback from mentors, and prioritizing tasks with a problem-solving approach.

Overall, the internship not only fortified my database and programming skills but also equipped me to meaningfully contribute to professional software development teams. I am now more confident in addressing difficult backend problems, following principles of clean code, and creating security conscious, high-performing systems that scale in real-world applications.

**ACKNOWLEDGEMENTS**

First, we indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Pro-VC - Engineering and Dean, Presidency School of Computer Science and Engineering & Presidency School of Information Science, Presidency University for getting us permission to undergo the project.

We express our heartfelt gratitude to our beloved Associate Dean **Dr. Mydhili Nair,** Presidency School of Computer Science and Engineering, Presidency University, and Dr. “Dr. ASIF MOHAMMED”, Head of the Department, Presidency School of Computer Science and Engineering, Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide **Dr. VIJAYA KUMAR A V, Professor** and Reviewer **Dr. NAVEEN KUMAR N M**, Presidency School of Computer Science and Engineering, Presidency University for his inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the internship work.

We would like to convey our gratitude and heartfelt thanks to the PIP4001 Internship/University Project Coordinator **Mr. Md Ziaur Rahman and Dr. Sampath A K,** department Internship Coordinators “**Mr. Md Ziaur Rahman**” and Git hub coordinator **Mr. Muthuraj.**

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

**TABLE OF CONTENTS**

**ABSTRACT………………………………………………. 05**

**ACKNOWLEDGEMENT……………….………………. 06**

**CONTENTS………………………………………………. 07**

**CHAPTER 1-INTRODUCTION………………...……….08**

**CHAPTER 2- LITERATURE SURVEY………….……...09**

**CHAPTER 3-PROPOSED SYSTEM/WORK…………...21**

**CHAPTER 4-PROBLEM STATEMENT………………..24**

**CHAPTER 5-OBJECTIVES…….………………….…….26**

**CHAPTER 6-DESCRIPTION OF DUTIES….………….29**

**CHAPTER 7- PROFICIENCIES….…………..………….31**

**CHAPTER 8-OUTCOMES…….…………………………33**

**CHAPTER 9-CONCLUSION.……………………………35**

**CHAPTER 10-ENCLOSURE…..………………………...36**

**Chapter 1**

**INTRODUCTION**

As an intern at L&T Technology Services, I had the enriching experience of being a part of the technological innovation and engineering excellence world. L&T Technology Services, a leading global engineering services provider, has emerged as a front runner in delivering innovative solutions and services to industries such as automative, aerospace, telecom and healthcare, among others.

Established in 2012, L&T Technology Services has earned an impeccable reputation for its focus on providing high quality engineering and R&D services that meet the changing needs of the digital age. With a deep focus on creating innovation and using cutting-edge technologies, the company has established itself as a partner of choice for companies looking to drive their digital transformation journey forward while ensuring exceptional quality and efficiency in their processes.

With its diversified portfolio of services covering product design, testing, and implementation of innovative technologies, L&T Technology Services has been instrumental in changing the face of engineering and advancing technologies in multiple sectors. Its objective is to offer end-to-end engineering solutions that help clients deliver their business goals while ensuring high quality, reliability, and sustainability.

Being a student intern for the first time, my experience at L&T Technology Services exposed me to a strong platform to observe practical details about the inner workings of the engineering field, developing my career as a professional and cultivating further knowledge of the intricacies and challenges that are contained in the technology and innovation field. This experience widened my exposure and gave me valuable skills and insights that will inform my future endeavors in technology and engineering.

**Chapter 2**

**LITERATURE SURVEY**

**1. RESTful API Development using FastAPI**

**Findings:**

REST (Representational State Transfer) is a widely accepted architectural style for building APIs due to its stateless nature, resource-oriented URLs, and compatibility with HTTP methods. The traditional frameworks of Flask and Django REST Framework are popularly applied in Python environments. Due to the increased requirement of asynchronous processing and high-performance web services, FastAPI has been developed as a new alternative.

**Observations:**

While Flask is mature and beginner-friendly, it does not have native support for type validation or asynchronous operations. This results in scalability problems and manual schema management. FastAPI, by contrast, provides async I/O handling with Python's async/await, automatic OpenAPI documentation generation, and built-in Pydantic-based request validation—features that greatly minimize development time and bugs.

**Outcomes:**

Throughout the internship, I made use of FastAPI to implement strong and async REST APIs. My key contribution was developing secure and scalable registration and authentication user endpoints with error handling and auto-documentation. I made use of FastAPI's dependency injection for clean and modular code, which was never utilized in the legacy systems in the organization.

**References:**

•FastAPI Documentation (https://fastapi.tiangolo.com/)

•Richardson, L., & Amundsen, M. (2013). RESTful Web APIs. O'Reilly Media.

**2. Secure Password Management with AES & bcrypt**

**Findings:**  
Secure management of user credentials is an essential aspect of contemporary web applications. Best practices are implemented in the form of cryptographic hash functions such as bcrypt for password storage and symmetric encryption like AES for data transmission security. Even with these tools, a lot of systems are breaking down due to lax implementation practices like hardcoding secrets, employing weak algorithms, or ignoring encryption altogether.

**Observations:**

Old practices in certain internal applications utilized plaintext storage of passwords or had no end-to-end encryption upon login. Neither did there exist sufficient logging and error handling for the operations involved with authentication. All such deficiencies introduce significant threats such as brute-force and credential leakage attacks.

Results

I applied AES encryption (mode CBC) in the frontend using CryptoJS and performed decryption on the backend through pycryptodome. After decryption, I securely hashed the password via bcrypt before it was stored within PostgreSQL. Such a multi-tiered security architecture guarantees passwords never being transmitted or stored in plain text, effectively improving the application's security profile.

References:

• PEP 458 – Secure PyPI downloads with cryptographic signing.

• OWASP Authentication Cheat Sheet (https://cheatsheetseries.owasp.org/)

**3. PostgreSQL for Transaction-Safe Data Handling**

**Findings:**  
PostgreSQL is a robust open-source relational database system with reputation for ACID compliance, support for advanced queries, indexing, and data types of JSON. PostgreSQL is widely utilized in web development but tends to be underutilized within small or quick prototyping environments.

**Observations:**

Prior to my effort, the internal system used either JSON files or unnormalized table structures for the user data, leading to redundancy and inefficient query performance. Also, the absence of schema validation or foreign key relationships raised the risk of invalid data.

**Consequences:**

I created normalized schemas in PostgreSQL to enable secure storage of users with unique constraints and role-based access models. By using ORM mapping and indexing, I dramatically enhanced data reliability and query performance. I also facilitated integration with FastAPI for clean separation of database logic and business logic.

**References:**

• PostgreSQL Official Documentation (https://www.postgresql.org/docs/)

• Elmasri, R., & Navathe, S. (2015). Fundamentals of Database Systems (7th ed.). Pearson.

**4. PEP8 and Clean Code Standards**

**Findings:**  
PEP8 is the standard style guide for Python code, ensuring readability, consistency, and cooperation. Too many development teams do not implement these standards, leading to unstructured, unmaintainable codebases.

Observations:

During my early development phases, my code was inconsistent in naming conventions, line lengths, and documentation. Initial feedback from mentors stressed the importance of clean, modular, and standard-compliant code.

**Outcomes:**

I followed tools like flake8, black, and pylint to check coding standards programmatically. Through PEP8 compliance at every stage of the project, I enhanced maintainability and readability of code so it is easily comprehensible to other developers while reviewing and adding functionality. It also made sure that the project is in conformation with requirements for production-grade code.

**References:**

•topic PEP8 – Style Guide for Python Code: https://peps.python.org/pep-0008/

•van Rossum, G. (Author of Python). "The Zen of Python" (<https://peps.python.org/pep-> 0020/)

1. **Introduction to Computing Technologies**

Computing technologies themselves have been completely revolutionized from initial hardware-focused systems to extensively integrated, distributed computing platforms driven by cloud infrastructures. The initial computers were constrained by low processing capability, rigid structure, and direct human intervention reliance. The early systems executed tasks independently with minimum abstraction between hardware and application logic.

In contrast, modern paradigms emphasize computing modularity, scalability, and virtualization. The convergence of high-performance hardware technologies, e.g., multi-core processors, solid-state drives (SSDs), and high-throughput networks—into advanced software architectures has enabled the development of complex, distributed systems that can handle heterogeneous workloads.

It is necessary to have a complete understanding of both the hardware and software components in creating, deploying, and managing contemporary computing systems. Hardware forms the computational core in terms of CPUs, GPUs, memory hierarchies, and I/O devices. Software combines these resources in the form of operating systems, compilers, middleware, and application frameworks.

Along with this, cloud computing has also delivered elastic provisioning of resources, infrastructure-as-code (IaC), and platform abstraction through service models like IaaS, PaaS, and SaaS. All these advancements have supported rapid deployment and scaling of applications, which form the basis of technologies like artificial intelligence (AI), machine learning (ML), edge computing, and containerization.

As computing systems become increasingly distributed and sophisticated, an understanding of low-level architectural components as well as high-level layers of abstraction is essential to IT professionals, developers, and engineers operating within today's technology landscapes.

**2. Hardware and Software Fundamentals**

**Hardware:** Hardware consists of the physical components of a computing system, including the Central Processing Unit (CPU), Random Access Memory (RAM), storage devices (HDDs, SSDs), and peripheral equipment like keyboards, displays, and network interfaces. Understanding hardware architecture—like instruction sets, cache hierarchies, buses, and input/output operations—is necessary for performance optimization and system programming.

Major books such as Computer Organization by Carl Hamacher provide detailed analysis of the workings of the computer machine from within, emphasizing aspects such as pipelining, memory management, and instruction cycles.

**Software:** System software (OS) and application software.

The contribution of compilers, interpreters, and development environments to software execution.

serves as a user interface to hardware. It consists of system software (operating systems, device drivers, file systems) and application software (browsers, text editors, business applications). Compilers and interpreters convert high-level programming code into machine-executable code, while Integrated Development Environments (IDEs) support effective software development with debugging, code completion, and project management features.

**3. Operating Systems:**

OS manages hardware resources, offers a user interface, and executes applications.

Common OS options: Windows, Linux, macOS.

**Key research:** OS evolution from Unix to contemporary Linux distributions and Windows OS developments.

An Operating System (OS) is tasked with abstracting and handling hardware resources like CPU scheduling, memory, and disk I/O. It serves to mediate between user programs and physical hardware for safe and effective operation.

Well-known operating systems are Windows, Linux, and macOS. Linux, among others, has become the workhorse of the majority of server and cloud ecosystems because it's open-source and supported by an enormous community of users. How the design of an OS developed—from monolithic kernels in Unix to modular and microkernel systems in modern computers—shows how scalability and dependability have come to be demanded.

**4. Introduction to Cloud Computing**

Cloud computing transforms IT infrastructure into scalable, on-demand resources available over the internet. Virtual machines, storage, and services are provisioned dynamically by users without physical hardware control.

Cloud services are classified as:

•IaaS (Infrastructure as a Service): Virtual servers, networking (e.g., AWS EC2).

•PaaS (Platform as a Service): Development platforms (e.g., Google App Engine).

•SaaS (Software as a Service): Web-based software (e.g., Google Workspace, Microsoft 365).

Virtualization abstracts hardware with hypervisors (e.g., VMware, KVM), and containerization (e.g., Docker, Kubernetes) separates applications for consistency between environments, enhancing deployment speed and resource utilization.

Reference: Cloud Computing: Principles and Paradigms by Rajkumar Buyya.

**5. Python as a Programming Language**

Python emerged as a popular programming language because of its clean syntax, readability, and rich standard library. It supports various programming paradigms—object-oriented, procedural, and functional.

•Web development (e.g., Flask, FastAPI, Django)

Python's extensibility and rich ecosystem make it an ideal language for quick development and prototyping.

**6. Development Tools: VS Code**

Visual Studio Code, or VS Code, is an open-source, cross-platform code editor used widely by developers because of its high performance and adaptability. It can support a vast majority of programming languages through extensions, with complete support for Python, JavaScript, C++, etc.

**Major features:**

•Built-in debugger and terminal

•IntelliSense (context-sensitive code suggestions)

•In-built Git version control

•Extension marketplace to add linters, formatters, Docker integration, etc.

Its light weight along with robust development tools make it perfect for full-stack and backend development processes.

**7. FastAPI and Uvicorn**

FastAPI is an evolved Python web framework for creating high-performance APIs. It's built on Starlette for web processing and Pydantic for data types, and it has built-in support for asynchronous programming with Python's async/await syntax, which results in non-blocking I/O operations.

**FastAPI includes:**

•Automatic OpenAPI and Swagger documentation

•Dependency injection built-in support

•Great performance in comparison to Node.js and Go-based APIs

**Uvicorn** is a web server that's ASGI-compatible and is utilized to execute FastAPI applications. It supports concurrent handling of requests via event loops, thus lending itself well to real-time applications and microservices.

FastAPI compared to Flask and Django provides more rapid performance, improved type checking, and async native support.

**Reference: FastAPI Official Documentation**

**8.PostgreSQL as a Relational Database System**

PostgreSQL is a robust, open-source object-relational database management system (ORDBMS) that is renowned for its reliability, feature set, and standards conformance. It provides support for both standard relational database features and enhanced capabilities such as JSON data types, full-text search, and user-defined functions.

**Key Features:**

•ACID compliance for transactional reliability

•Sophisticated indexing methods (B-tree, GIN, GiST, BRIN)

•Extensibility via custom types, operators, and procedural languages (PL/pgSQL, PL/Python)

•Concurrency control through Multi-Version Concurrency Control (MVCC)

• Robust security with roles, fine-grained privileges, and SSL support

PostgreSQL is well-used in enterprise and cloud environments because it supports horizontal scaling (through logical replication or tools such as Citus) and fully integrates with contemporary ORMs such as SQLAlchemy.

PostgreSQL is most used these days with Python-based frameworks such as FastAPI to give it a strong backend stack. Being used together helps it to make effective use of structured data, transactional consistency, and efficient queries in high-performance REST and asynchronous applications.

**Use Cases:**

•Complex transactional systems (such as finance and inventory)

•Reporting and analytical queries

•Hybrid data models where structured and semi-structured data exist

**Reference: PostgreSQL Official Documentation**

**9.Python Coding Standards: PEP8 Guidelines**

PEP8 (Python Enhancement Proposal 8) is the standard style guide for writing clean, readable, and consistent Python code. It defines conventions that ensure code quality between teams and projects, particularly in collaborative or production-grade software development environments.

**Major PEP8 Guidelines:**

•Indentation: Indent using 4 spaces per indentation level.

•Line Length: Keep all lines to a maximum of 79 characters to improve readability.

• Blank Lines: Use blank lines to distinguish functions, classes, and blocks of code within functions.

• Imports: Organize imports into three groups (standard libraries, third-party libraries, local application imports) and put each group on a separate blank line.

• Naming Conventions:

o Variables/functions: snake\_case

o Classes: CamelCase

o Constants: UPPER\_CASE

• Whitespace Usage:

o Avoid unnecessary whitespace in expressions and statements.

•Comments: Inline comments should be used sparingly and block comments to comment complex logic. Docstrings ("\"\"Docstring\""") for modules, classes, and functions.

Tools Supporting PEP8:

•flake8: Lint tool that verifies compliance with PEP8 and marks potential problems.

•black: Auto-formatter that enforces uniform code style based on a strict subset of PEP8.

•pylint: Does detailed static code analysis and offers recommendations on naming, formatting, and performance.

Conformity to PEP8 guarantees improved maintainability, less stylistic-consistency-related bugs, and enhanced code readability—particularly of utmost importance in large codes or open-source code contributions.

**Reference: PEP8 - Style Guide for Python Code (**[**https://peps.python.org/pep-0008/**](https://peps.python.org/pep-0008/)**)**

**10.RESTful API Design Principles**

REST (Representational State Transfer) is a networked application design architectural style, specifically APIs. RESTful APIs are stateless, scalable, and rely on standard HTTP methods to operate on resources, which makes them suitable for web services and contemporary microservice architecture.

**Fundamental REST Principles:**

•Statelessness: Every client request must include all the information needed. The server never stores any session information between requests.

•Resource-Based: Everything is a resource, and each resource has a unique URI. Resources are normally represented in JSON or XML.

No Example: /users/123, /products/45/reviews

**•HTTP Methods:**

GET: Fetch a resource

POST: Create a new resource

PUT: Update/replace an existing resource

PATCH: Partially update a resource

DELETE: Delete a resource

•Stateless Communication: The server doesn't store client context; all the relevant information must be in the request (typically through headers or the body).

•Representation: Resources may be represented in various formats (e.g., JSON, XML), enabling flexible client-server communication.

•HATEOAS (Hypermedia As The Engine Of Application State): A mature REST principle where the server returns links to lead clients through available actions dynamically.

Best Practices in RESTful Design:

•Use plural nouns to name resources (e.g., /users, /products).

•Keep hierarchical structure for nested resources (e.g., /users/123/orders).

•Use HTTP status codes correctly:

200 OK for successful requests

201 Created on successful POST

04 No Content on successful DELETE

400 Bad Request on client-side errors

401 Unauthorized or 403 Forbidden on authentication/authorization errors

404 Not Found when a resource does not exist

**REST in FastAPI:**

FastAPI inherently supports RESTful concepts by converting Python functions to HTTP routes and methods. With asynchronous I/O support, automatic validation of data with Pydantic, and OpenAPI schema generation, it supports rapid development of solid RESTful services.

**Integration with PostgreSQL:**

Applying REST concepts to a relational database such as PostgreSQL enables:

•Uniform CRUD operations directly mapped to HTTP methods

•Persistence and integrity of data through transactional support

•Filtering, sorting, and pagination of big data in query parameters (i.e., /users?page=2&limit=50)

**11.User Management and Authentication Systems**

Contemporary web applications demand strong user management and authentication mechanisms to maintain safe access control, customized user interfaces, and data protection compliance.

User Management

User management entails the creation, modification, and removal of user accounts, usually maintained in a relational database like PostgreSQL. Important functionalities are:

•Account registration: Captures user information and securely stores it.

•Profile management: Enables users to update personal information and preferences.

•Role-based access control (RBAC): Defines various permissions for roles (e.g., admin, user, guest) to limit access to sensitive endpoints.

•Password management: Enforces secure password policies, supports password reset, and stores passwords securely through hashing algorithms such as bcrypt.

Authentication Systems

Authentication authenticates a user's identity prior to granting access to protected resources. The most prevalent methods include:

• Basic Authentication: Username and password are included in each request (use with caution, without HTTPS).

• Token-Based Authentication:

**JWT (JSON Web Tokens):** Encodes user data into a signed token clients persist and include with every request.

Advantages: Stateless, scalable, and supports expiration claims and embedded metadata.

Example: FastAPI supports JWT authentication through libraries such as python-jose or PyJWT.

OAuth 2.0: Enables users to authenticate through third-party providers such as Google, GitHub, or Facebook, helpful for social login integrations.

Multi-Factor Authentication (MFA): Introduces a second factor (e.g., OTP, biometric authentication) to provide added security.

**Implementation in FastAPI**

FastAPI provides a modular and secure way to implement authentication using:

•Dependency injection for security checks

•OAuth2PasswordBearer for token-based login flows

•bcrypt for password hashing and verification

•JWT utilities for creating and verifying tokens

**Example login flow:**

1.User submits credentials via /auth/login

2.Credentials are verified and a JWT token is generated

3.The token is sent in the Authorization header (Bearer <token>) on future requests

**Security Considerations**

•Always store passwords hashed and salted (e.g., using bcrypt)

•Enforce rate-limiting to stop brute-force attacks

•Utilize HTTPS to secure data in transit

•Sanitize and validate all user input to stop SQL injection and XSS attacks

•Regularly log and audit login attempts and suspicious activity

User management and authentication systems are critical to secure access and personalized user experiences in RESTful web applications. Coupled with frameworks such as FastAPI and databases such as PostgreSQL, they form a sound basis for developing scalable, secure backends.

**12.Conclusion**

The development of computing technologies has introduced a dramatic change in the way applications are created, deployed, and maintained. Starting from the basic knowledge of hardware and software basics, and moving on through the function of operating systems, this report investigated how the underlying infrastructure supports all layers of contemporary computing.

With the introduction of cloud computing, environments such as AWS and Azure have provided on-demand scalability, while technologies like virtualization and containerization (e.g., Docker, Kubernetes) have improved resource utilization and deployment consistency.

At the coding level, Python has become a ubiquitous language because it is readable, versatile, and rich in an ecosystem—suitable for jobs from automation to web development at all levels. Compliance with PEP8 conventions also guarantees code quality, readability, and ease of maintenance.

The inclusion of Visual Studio Code as a core development tool, complete with debugging, version control, and extension support, greatly enhances developer productivity. Combined with FastAPI—a modern, high-performance web framework—and Uvicorn—a high-powered ASGI server—developers are now able to create asynchronous, production-quality APIs with ease.

The backends for these APIs are usually fueled by strong relational databases such as PostgreSQL, which provide scalability, transactional consistency, and rich querying support. These technologies are then structured using RESTful design patterns to ensure that systems are modular, scalable, and easy to integrate with.

Finally, secure and scalable user management and authentication systems provide the foundation of contemporary applications, supporting secure access control, data protection, and customized user experiences. Through practices like JWT-based token authentication, role-based access control, and bcrypt-encrypted password storage, applications can maintain both usability and security.

Collectively, this toolbox of principles, technologies, and tools forms the complete basis upon which secure, scalable, and efficient modern software systems can be constructed. Mastery and application of these elements are necessary for any would-be software developer or systems architect working through the present and future landscape of cloud-native and API-driven applications.

**Chapter 3**

**PROPOSED SYSTEM / WORK**

As part of my internship, I was tasked to work on helping develop web application in-house project. The project was to develop a secure, modular, and scalable user management and authentication system that would serve as the core of a more extensive security and monitoring platform. The system was developed based on RESTful architecture, built with FastAPI, and supported by a solid PostgreSQL database. The objective was to bring the development process in accordance with contemporary backend practices and development standards of the industry.

**First Phase** – Foundation Building and Learning

During the initial weeks of the internship, I concentrated on learning the basic concepts of computing technologies, which served as a foundation for subsequent development activities. These included:

•Fundamentals of hardware and software: Learning CPUs, memory, storage, peripherals, and the functions of system vs. application software.

•Operating Systems: Investigation of OS design, process scheduling, memory management, and applicability in server landscapes (Windows or Linux).

•Cloud Computing: Research on cloud deployment models (IaaS, PaaS, SaaS), environment for deployment, and technology such as Docker, Kubernetes.

•Programming with Python: Learning of Python syntax, control structures, data types, and incrementally constructing object-oriented programming concepts and libraries.

•Version Control: Getting introduced to Git as a means for code control, collaborative work, and version history using GitHub.

•Setup of Tools: Visual Studio Code setup, setup of PostgreSQL client tools, and virtual environments to support Python programming.

Development Stage in house Implementation of Project

After the technical groundwork was completed, I started actively working towards developing the in-house project user registration and authentication module, practicing modern backend engineering principles

**Tech Stack Overview:**

• FastAPI: For building RESTful APIs with async support.

• Uvicorn: ASGI server to run FastAPI apps efficiently.

• PostgreSQL: Used as the core database for storing user information.

• bcrypt: For secure password hashing and verification.

• JWT (JSON Web Tokens): Employed for stateless authentication as well as secure session management.

• CryptoJS (Frontend): Used by the frontend to AES (CBC mode, ZeroPadding) encrypt passwords before transmission.

• Python-JOSE: To manage JWT token generation and validation in the backend.

• Python-dotenv: For safe management of environment variables (e.g., DB credentials, secret keys).

**Core Features Implemented:**

**1. User Registration (/auth/registerUser):**

• Receives password input from frontend in AES encryption.

• Decrypts the password in the backend using pycryptodome with a pre-established key and IV (1234567812345678).

• Checks for input (username/email/password), duplicate in PostgreSQL.

• Hashes the decrypted password using bcrypt before storage.

• Logs registration event for auditing.

• Returns proper HTTP response based on outcome (success/failure).

**2. Authentication and Token Generation (/auth/login):**

• Compares hashed password stored in database with input.

• Creates JWT tokens upon successful login, including user ID and expiration metadata.

• Implements stateless session management by requiring the token for access to protected routes.

• Follows OAuth2PasswordBearer flow for protected endpoints.

**3. Error Handling and Logging:**

• Employed structured error messages using HTTPException and status codes.

• Provided descriptive log messages for successful and failed operations using Python's logging module.

• Made sure that sensitive data (e.g., raw passwords) were never logged or exposed.

**Security and Best Practices Implemented:**

• Implemented PEP8 standards to write clean, readable Python.

• Used dependency injection to improve testability and reusability in FastAPI routes.

• Kept secret keys and database URLs in environment variables (loaded with.env files).

• Sanitized all the inputs to prevent SQL injection and logic errors.

• Implemented JWT expiration time and manual revocation handling for token security.

• Implemented RESTful conventions on endpoints (POST /registerUser, POST /login, GET /me).

**Challenges Faced & Lessons Learned:**

• My first implementation wasn't firmly structured and got rejected by my mentor. Then I came to know about service-repository separation, modularity in code, and clean architecture.

• I struggled a lot with AES decryption logic at first, especially bringing the frontend's CryptoJS configuration in Python. Debbuging byte padding and key/IV utilization, I finally implemented successfully functional decryption logic.

•Learning how to assemble all the components—database, validation, token logic, and encryption—to create something useful was instrumental in grasping how intricate backend systems are assembled in practical applications.

•I loved working on issues that make logs, error handling, and edge-case handling necessary in production-grade applications.

**Summary of Contribution**

This internship helped me mature from a Python and backend programming beginner to someone who can design and protect real-world APIs using FastAPI and PostgreSQL. I gained a hands-on expertise:

•API creation

•database design

•security and cryptography

•code structure and readability

•RESTful guidelines

•token-based authentication

These learnings and contributions in addition created the C-Watch user management system and laid the groundwork for more complex modules in the future.

**Chapter 4**

**PROBLEM STATEMENT**

**1.Task Overview:**

Build a RESTful API using FASTAPI for user management (CRUD operations) using file-based

storage or DATABASE (e. g., JSON) .

**Requirements:**

**1.API Development:**

Develop the following endpoints: POST /usersCreate a new user.

GET /users Retrieve all users Response: List of users from the JSON file.

GET /users/<id>Retrieve details of a specific user.

PUT /users/<id>Update user details.

DELETE /users/<id>Delete a user.

**2.File-Based Storage:**

Use a JSON file to store user data (no database).

**2.Task overview:**

Please find your problem statement:

You are building a Document Management System (DMS) that supports version control, access control, and different document types (Text, PDF, and Code files). The system should be able to track document changes and allow multiple users to interact with documents securely.

**Requirements:**

Create a base class Document

* Attributes: title, author, \_content (protected), version
* Methods:
  + edit\_content(new\_content): Updates content and increases version.
  + display\_content(): Displays content.
  + get\_version(): Returns the current version.
  + \_update\_version(): A private method to increment version.

Create a derived class TextDocument

* Additional Attribute: word\_count (calculated dynamically).
* Override edit\_content() to update word\_count.

Create another derived class PDFDocument

* Additional Attribute: encryption\_status.
* Override edit\_content() to prevent editing if encrypted.
* Add encrypt\_document() to toggle encryption.

Create a derived class CodeDocument

* Additional Attribute: language (Python, Java, etc.).
* Override display\_content() to apply syntax highlighting (mock implementation).

Create a user class

* Attributes: username, role ("admin", "editor", "viewer").
* Methods:
  + edit\_document(document, new\_content): Only "admin" and "editor" can edit.
  + view\_document(document): Anyone can view content.

**Expected Challenges & Tricks:**

1. Encapsulation: Prevent direct modification of \_content and \_version (use getters & setters).
2. Polymorphism: display\_content() behaves differently for TextDocument, PDFDocument, and CodeDocument.
3. Access Control: Users can only edit if they have the right role.
4. Version Control: The system tracks change automatically.
5. Private Methods: \_update\_version() should not be accessed outside the class.

**Chapter 5**

**OBJECTIVES**

The overall goal of the internship was to develop hands-on experience in constructing a technically solid and secure User Management and Authentication System as a part of an in-house software development project. The project would simulate real-world development practices and ready interns for effective contribution towards full-stack backend systems with up-to-date technologies. The below-mentioned goals were pursued during the internship tenure:

**1. Learning Objectives**

These goals were aimed at gaining basic technical understanding and gearing up for active development:

•Build Core Computing Foundations:

Learn how hardware components such as CPU, memory, storage, and peripheral devices work.

Learn about software components such as system software (e.g., operating systems) and application software.

Learn about the role of compilers, interpreters, and development environments in running programs.

•Investigate Operating Systems and Server Environments:

Understand the fundamental tasks of an OS: process management, memory management, I/O handling, and file systems.

Study well-known operating systems like Windows, Linux, and macOS and why they are important today in computer environments.

• Understand Cloud Computing Concepts:

Study cloud service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

Understand virtualization and containerization technologies (e.g., Docker, Kubernetes) and their application in scalable application deployment.

•Develop Proficiency in Python Programming:

Master core concepts such as data types, functions, control flow, and object-oriented programming.

Get familiar with leading Python libraries and packages useful for web development, encryption, and API communication.

•Set Up and Utilize Development Tools:

Install and set up Visual Studio Code (VS Code) for backend development using Python.

Utilize features like integrated terminal, linting, debugging extensions, and version control integration.

Recognize the significance of version control with Git and GitHub for collaborative development.

•Know RESTful API Design Principles:

Know how RESTful APIs adhere to resource-oriented structure.

Know HTTP methods (GET, POST, PUT, DELETE), status codes, and data handling through request/response patterns.

**2. Development Objectives**

These objectives were aimed at deploying major features of the in-house system with modern backend technologies and standards:

•Build a Secure User Registration System:

Take and decrypt AES-encrypted passwords sent from the frontend with a fixed key and IV (coinciding with AES CBC mode and ZeroPadding).

Validate input and avoid duplicate entries of users.

Securely store passwords in a PostgreSQL database using bcrypt for hashing.

See to it that sensitive information is never logged or revealed during the process.

•Install Login and Authentication Using JWT

Validate users using saved credentials and give JWT tokens when login is successful.

Make token secure by adding expiry metadata and deal with token verification on subsequent requests.

Make endpoints secure using OAuth2 password flow and dependency injection.

•Create a Modular and Scalable Backend Architecture:

Organize the codebase into layers that have significance like API routes, services, schemas, database, constants, and security helpers.

Apply object-oriented concepts to enclose business logic and enable extensibility in the future.

•Ensure Code Quality Using PEP8 Standards:

Write clean, readable Python code that adheres to industry-standard conventions.

Use linters and formatters to maintain consistency and clarity across modules.

•Integrate PostgreSQL for Persistent Data Management:

Create efficient schemas and queries to manage user data reliably.

Use connection pooling and parameterized queries to prevent SQL injection and performance bottlenecks.

•Enable Logging and Error Handling Mechanisms:

Implement structured error messages for predictable client responses.

Use Python's logging module to track critical operations, warnings, and errors.

• Follow RESTful Design Principles Across All Endpoints:

Maintain consistent endpoint naming conventions.

Use appropriate HTTP methods and status codes.

Provide clear separation of responsibilities among request handling, business logic, and database access.

• Support Future Growth and System Scaling:

Keep the backend modular to make it easy to add features such as role-based access control, user profile management, and password recovery.

Make architectural choices (e.g., token-based authentication, service layer abstraction) so that they enable future scalability and maintainability.

**Chapter 6**

**DESCRIPTION OF DUTIES**

During the first few weeks of my internship, I was concentrating on understanding the basic principles of Python programming, backend development, and API handling. As my Python skills were not good at the beginning, my mentor gave helpful advice and guidance, and explained to me the basic concepts so that I could enhance my coding skills. I invested time in learning FastAPI, MongoDB, and PostgreSQL, which are essential for backend system construction and data management. I also learned about RESTful design strategies and how to effectively structure backend applications. This stage gave me a solid technical background for the responsibilities and tasks that lie ahead.

My manager was also instrumental in opening my eyes through exposure to the hardware aspect of the company. I realized how the build-out of backend systems relates to physical hardware infrastructure, and how the two interact in practice applications. I also had some time with the company's manager, who personally walked me around the company labs, where I gained knowledge regarding the various tools and technologies the company employs as part of its hardware development work. This exposure taught me about the overall working culture of the organization and how the hardware and software teams work together. The mix of technical learning and understanding of the culture of the organization facilitated my quick acclimation to the professional world.

After I had a strong grasp of the fundamentals, I was given individual tasks to implement major backend features like developing rest APIs, managing database operations, and coding reusable modules. These tasks not only gave me hands-on practice with actual development approaches but also enabled me to delve deeper into more complex backend concepts. I was assigned to develop API endpoints to execute CRUD operations, integrating APIs with PostgreSQL and MongoDB, and organizing the code in a modular fashion to provide scalability and reusability. Moreover, I was also tasked with implementing security features like input validation and error handling.

As I finished every task, my mentor would review my code and give me detailed feedback that I would incorporate to enhance the logic, structure, and performance of my code. Learning how to refine my development habits through this feedback improved my ability to write clean and maintainable code. I learned the value of code reviews in team development and how to continuously improve software. This step helped considerably improve my ability to solve problems and increase confidence in shifting between fundamental exercises to applying production-standard code with realistic standards and needs.

Upon transferring to the solution project, I had work compartmentalized in terms of manageability, whereby every module was handed to me sequentially by the mentor. This approach allowed me to focus on one specific functionality at a time and gain a deeper understanding of the project. My first attempt at writing the code for the project was rejected due to its basic approach, lack of scalability, and insufficient depth. This feedback was precious, as it pointed out where I needed to make improvements, specifically in how I could use more effective algorithms and organize the code in a more modular and maintainable manner.

Upon reviewing the feedback closely, I made the changes and perfected my code to the requirements of the project. I paid closer attention to the best practices regarding performance, security, and maintainability. Every module that I did, whether it was for user sign-up, login, handling of encrypted data, or backend integration, helped me hone my craft and use more sophisticated techniques. With every pass, I gained more confidence in creating stable and scalable systems following both functional as well as non-functional requirements. This experience not only allowed me to develop my backend development skills but also imparted to me the value of code reviews and ongoing improvement in actual software development.

During the final weeks of my internship, I kept working on the solution project, enhancing important backend elements, integrating various aspects of the system, and improving my code based on feedback from mentors. The exercises I performed enabled me to improve my knowledge in backend development, especially in user management and authentication systems. The hands-on experience, coupled with the iterative feedback process, significantly contributed to my development as a backend developer and readiness for challenges ahead.

**Chapter 7**

**PROFICIENCIES ENHANCED**

Through the internship experience, I gained considerably in my technical capabilities from working on the solution project. The project offered exposure to vast practices of backend development, including the development of secure APIs as well as using databases and handling authentication. Through the internship experience, I obtained thorough knowledge about key technologies and methods that make up the essential requirements of today's backend systems while enjoying practical mentorship and hands-on expertise.

**Technical Skills**

**1.Python Programming**

Acquired the skill of coding efficient, clean, and maintainable Python code.

Better knowledge of Python fundamentals like data structures, functions, classes, and modules.

**2.FastAPI Framework**

Understood how to create fast and stable web APIs with FastAPI.

Got a good grasp of RESTful API basics and how to design and manage API endpoints for user management.

**3.Database Management (PostgreSQL & MongoDB)**

Acquired hands-on experience with PostgreSQL for relational database management and MongoDB for NoSQL data storage.

Acquired skills in CRUD operations, data modelling, and optimizing database queries.

**4.User Authentication & Security**

Implemented secure user authentication mechanisms using JWT tokens and bcrypt password hashing.

Acquired knowledge on how to encrypt and securely store sensitive data using AES encryption and decryption techniques.

**Soft Skills**

**1.Collaboration**

Worked in a team environment, collaborating with my mentor and other developers.

Developed skills in dealing with feedback and adjusting enhance the overall quality of your code.

**2.Problem Solving**

Faced several challenges during the project, such as dealing with encryption and incorporating complex authentication mechanisms.

Developed skills in tackling problems systematically, identifying solutions, and implementing them into real-world applications.

**3.Time Management**

Handled multiple tasks and deadlines during the internship, learning how to prioritize work and deliver project needs efficiently.

**4.Continuous Learning**

Continuously developed technical skills by way of mentorship and self-improvement.

Acknowledged further insight into backend systems, security concepts, and API design, which will prove vital in subsequent development opportunities.

As a part of my internship responsibilities, my participation in daily sync-up meetings provided me with valuable insights into effective communication, project management, and team collaboration. Scribing numerous meetings facilitated the development of my active listening and note-taking skills, enabling me to accurately capture and summarize critical discussions, action items and project milestones.

This exposure also reinforced my flexibility and ability to work in collaboration in a dynamic and aggressive working environment, demonstrating my potential to contribute substantially to project deliverables and team goals.

Above all I learned to work in a team, how to approach people with the proper knowledge and ask them for their opinions so that we reach a decision quickly and effectively.

**Chapter 8**

**OUTCOMES**

The internship was a valuable experience to implement a live in-house project, providing hands-on exposure with contemporary backend development technologies. The results can be generally categorized as technical skills acquired, project outcomes, and career growth.

**1. Technical Skills Acquired**

•Comprehensive Knowledge of Software and Hardware Principles:

Acquired basic understanding regarding computer structure, operating systems, and their interface with application programs to execute contemporary computing platforms.

•Familiarity with Python Programming:

Acquired robust Python programming skills with an emphasis on object-oriented concepts, modularization of code, encryption methods, and secure storage of sensitive information.

•RESTful API Development with FastAPI:

Understood how to create, build, and document high-level REST APIs using FastAPI, such as the use of asynchronous programming and automatic OpenAPI generation.

•Implementation of Authentication Systems:

Created a safe login system that makes use of AES encryption for password passing, bcrypt hashing for secure storage, and JWT-based token authentication for session handling.

•Backend Architecture Design:

Learned how to organize a scalable backend project through service layers, schema validation, environment configurations, and modular directories in order to maintain separation of concerns.

•PostgreSQL Integration:

Created and integrated a relational database with the use of PostgreSQL, composing queries and employing ORM tools as required for secure data storage and retrieval.

•Adhering to PEP8 Standards and Clean Code Practices:

Confirmed that all Python code complied with industry standards' PEP8 formatting principles, making it more readable and easier to maintain.

•Mastering Tools: Visual Studio Code & Git:

Leveraged VS Code efficiently for development, utilizing in-built tools such as linters, terminal, and debuggers. Also experienced version control and collaboration with Git and GitHub.

**2. Project Deliverables**

•User Registration API: Developed a secure, validated endpoint for registering new users, managing encrypted passwords and storing credentials in the database.

•Login and JWT Token Generation: Set up token-based login that authenticates users and returns a JWT to gain access to protected resources.

•Secure Password Handling: Utilized AES (CBC mode) for decrypting front-end-encrypted passwords and securely hashed them prior to storing with bcrypt.

• Scalable Project Structure: Established an efficient project structure with well-defined roles for every module, ensuring reusability and extensibility.

• Logging and Debugging Setup: Implemented a logging facility for error monitoring and debugging, ensuring easier maintenance.

**3. Professional and Personal Growth**

• Enhanced Problem-Solving Skill:

Faced and resolved actual real-world technical challenges, such as encryption mismatches, async processing, and database integration issues.

• Real-World Development Workflow Understanding:

Learned to decompose features into smaller modules, review code in iterations, and learn from mentor feedback.

•Industry Tools and Practices Confidence:

Acquired hands-on experience with tools and practices applied in production environments, creating a solid foundation for future software development careers.

•Team Communication and Mentorship Interaction:

Learned the value of technical mentorship, writing down work, and frequent communication of progress in a professional team environment.

**Chapter 9**

**CONCLUSION**

My internship experience at L&T Technology Services was a learning and enriching experience that played a pivotal role in shaping my technical capabilities as well as professional attitude. This experience gave extensive real-world backend development exposure, an in-depth learning in Python programming, API design, user authentication, and secure data storage.

During the initial phase of the internship, I emphasized learning the fundamentals of computing — hardware, operating systems, and cloud technologies — to get an understanding of the larger picture of software systems. As the internship went on, I used these principles to design a secure, scalable, and modular user management and authentication system as part of an in-house solution project.

While developing using FastAPI, PostgreSQL, and libraries such as Uvicorn and VS Code, I implemented core backend features like user registration, hashed password management, JWT authentication, and appropriate logging and error handling. The application adhered tightly to PEP8 standards and RESTful API design principles in order to meet industry-level coding standards.

The learning curve was steep, especially since I started with minimal experience in Python and backend frameworks. Nonetheless, regular mentorship and incremental feedback enabled me to overcome initial setbacks, improve my coding habits, and become confident in managing intricate backend logic.

From AES decryption and secure login flows implementation to database integration and coding for scalability, each project reinforced my problem-solving skills and technical proficiency. I also learned valuable soft skills like collaboration, flexibility, and effective communication—important traits in any professional growth setting.

This internship has not only given me the technical expertise to succeed as a backend developer but also instilled in me the mindset to keep learning and growing. The experiential learning, coupled with actual challenges faced in real-time and mentor guidance, has readied me to be able to make significant contributions towards future development work and strive for excellence in the constantly changing scenario of software technology.

I am deeply thankful to L&T Technology Services for this precious experience and eagerly anticipate

being able to capitalize on this exposure as a pillar for my future work in the field of software development.

**Chapter 10**

**ENCLOSURES**

**3. Similarity Index / Plagiarism Check report clearly showing the Percentage (%). No need for a page-wise explanation.**

**SUSTAINABLE DEVELOPMENT GOALS (SDGs):**

****

**The project work carried out here is mapped to the below 2 goals:**

**Goal 8: Decent work and economic growth**

**Goal 9: Industry, Innovation and Infrastructure**