A TECHNICAL REPORT ON STUDENTS INDUSTRIAL WORK EXPERIENCE SCHEME(SIWES)

UNDERTAKEN AT

4, BALARABE MUSA CRESCENT, VICTORIA ISLAND, LAGOS, NIGERIA.

BY

IDOGUN JOHN OWOLABI
CPE/15/2418

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS OF THE BACHELOR'S DEGREE IN COMPUTER ENGINEERING

Department of Computer Engineering
Federal University of Technology, Akure

January 23, 2020

Certification

This is to certify that this report is a detailed account of Students' Work Experience Scheme (SIWES) undertaken by Mr. JOHN OWOLABI IDOGUN with Matriculation Number CPE/15/2418 for a period of six months at *ip*NX Nigeria Limited, Victoria Island, Lagos State and has been compiled in accordance to regulations guiding the preparation of reports in the Department of Computer Engineering, Federal University of Technology, Akure, Ondo State.

SIWES Coordinator's Signature

Head of Department's Signature

Industry-based supervisor's Signature

Student's Signature

Dedication

To my wonderful Guardians, benefactors and sponsors, Engr. & Mrs. Sunday Idogun: know I cherish the lessons you taught me which make me believe that with God, smart work, patience, and determination, I can achieve anything.

To my boss, Oluboyo Charles Oluwaseun: you are one of a rare kind.

To all techies: we have got a lot to learn.

Idogun John O.

Acknowledgements

I owe a huge debt of gratitude to God Almighty for His unconditional and resilient love cum compassion towards me. I appreciate him for the intellectual acumen He amply bestowed on me before, during and after this industrial training.

This project based internship was made possible by funding from my Guardians, benefactors and sponsors, Engr. & Mrs. Sunday Idogun, my amiable sister, Mrs. Modupe Igbekoyi, and her husband, Mr. Idowu Igbekoyi, my great brother, Mr. Festus Idogun, and all my family members. You have all been awesome to me. I would like to thank my industry based supervisor, Oluboyo Charles Oluwaseun, for his encouragement, supports in all aspects, and timely help. I am favoured to have been under your tutelage. Special thanks are extended to Frances Nnamadim, a Business Analyst and User Interface (UI)/User eXperience (UX) designer, who never seemed to be tired of imparting knowledge and writing comments and remarks, for discussions, feedback, ideas, and for help on the various projects done. I would also like to thank Amuche BensonOnyeibor, who extended her generosity and was kind enough to offer comments and suggestions while implementing some projects. My grateful thanks go to all the people at *ip*NX Nigeria Limited for the pleasant cooperation and hospitality they offered during my stay, particularly Shade EfiongBassey, the Human Resources Managers, and Administrators.

Finally, I want to thank Dr. Erastus Olarewaju Ogunti and Dr. (Mrs) Folasade Mojisola Dahunsi, Ag. Head Of Computer Engineering Department, Federal University of Technology, Akure, for their unpredented career advice and efforts to help secure internship placement.

Abstract

This paper reports a six-month Internship with the Research and System Architecture department at *ip*NX Nigeria Limited, Victoria Island, Lagos.

My tasks were to engineer, develop and deploy customer-centric software products while employing modern Software development paradigms and practices. Some projects require data visualisation in which communication between Front-end and Back-end asynchronously JavaScript Object Notation (JSON) over eXtensible Markup Language and HyperText Transfer Protocol Request (XHR) using Asynchronous JavaScript and eXtensible Markup Language (XML) (AJAX), while others involve implementing complex algorithms such as the Materialised Path Algorithm to provide some required complex features as contained in the Software Requirement Specification document. Such implemented complex features include but not limited to a Threaded Commenting System, Full-text search system using PostgreSQL's full-text search engine, a live chat system and other basic Create, Retrieve, Update and Delete (CRUD) processes using Flask and Django frameworks as well as PostgreSQL and SQLite databases. Most of the software projects were wholly written in Python, an interpreted, high-level, general-purpose programming language created by Guido van Rossum and first released in 1991 that lets one work quickly and integrate systems more effectively. Few other projects such as a Web crawler were implemented in Julia, a new programming language offering a unique combination of performance and productivity that promises to change scientific computing, and programming in general. Julia picks the best parts of existing programming languages, providing out-of-the-box features such as a powerful Read-Execute-Print Loop (REPL), an expressive syntax, Lisp-style meta-programming capabilities, powerful numeric and scientific programming libraries, a built-in package manager, efficient Unicode support, and easily called C and Python functions.

System/Linux Administration works were also embarked upon ranging from deploying already developed applications to a Linux server using Apache2 and nginx to complying with versioning and back-end compatibility of resources. In cooperation with a notable senior staff in the department of Business Intelligence and Data Analytics (BIDA), I designed and developed from scratch a complex blogging web application primarily to bring together tech enthusiasts. Syntax highlighting was implemented using Primejs and Django-ckeditor was used as "What You See Is What You Get (WYSIWYG)" rich texts or documents editor.

Contents

List of Figures

List of Tables

List of Abbreviations

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

CRUD Create, Retrieve, Update and Delete

DTL Django template language

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

JSON JavaScript Object Notation

ORM Object Relational Mapper

REPL Read-Execute-Print Loop

SDLC Software Development Life Cycle

SIWES Students' Work Experience Scheme

SQL Structured Query Language

SSH Secure Shell

UI User Interface

URL Uniform Resource Locator

UX User eXperience

vCPE Virtual Customer Premises Equipment

XHR eXtensible Markup Language and HyperText Transfer Protocol Request

XML eXtensible Markup Language

XHTML eXtensible HyperText Markup Language

XSLT eXtensible Stylesheet Language Transformations

WSGI Web Server Gateway Interface

WYSIWYG What You See Is What You Get

Chapter 1

Introduction to SIWES Program

1.1 Background

The SIWES is an undergraduate training programme with specific learning and career objectives geared towards the development of occupational and industrial competencies of the participating students. It is a requirement for the award of degrees and diploma to all students of tertiary institutions in Nigeria pursuing courses in specialised engineering, technical, business, applied sciences and applied arts. The scheme is a three-party program which involves the student, the tertiary institution, and the industry,and it exposes students to the practical knowledge of their course of study. Furthermore, SIWES is a general programme cutting across over 60 programmes in the universities, over 40 programmes in the polytechnics and about 10 programmes in the colleges of education. Thus, SIWES is not specific to any one course of study or discipline. Consequently, the effectiveness of SIWES cannot be looked at in isolation with respect to a single discipline, hence it is better explored in a holistic manner since many of the attributes, positive outcomes and challenges associated with SIWES are common to all disciplines participating in the scheme. Hence, the approach of this report is to look at SIWES as a general study programme cutting across several disciplines. Furthermore, the report gives details of the industrial work experience gained at *ip*NX Nigeria Limited, Victoria Island, Lagos state.

1.2 Brief History of SIWES Program

The SIWES started with about 748 students from 11 institutions of higher learning in 1974. By 1978, the scope of participation in the scheme had increased to about 5,000 students from 32 institutions. The Industrial Training Fund, however, withdrew from the management of the scheme in 1979 owing to problems of organizational logistics and the increased financial burden associated with the rapid expansion of SIWES. Consequently,

the Federal Government funded the scheme through the National Universities Commission (NUC) and the National Board for Technical Education (NBTE) who managed SIWES for five years (1979 – 1984). The supervising agencies (NUC and NBTE) operated the scheme in conjunction with their respective institutions during this period

1.3 Scope of SIWES

The decree which legislates this scheme does not give a time confinement or restriction to its implementation but it is a relative assignment. This implies that each school has the mandate with respect to her academic calendar to categorically state when the scheme will be undertaken but must ensure that the required period of training is satisfied thus.

1.4 Aims and Objectives of SIWES

The objectives of SIWES as stated in Information and Guideline for SIWES (2002) are:

1. To provide an avenue for students in higher institutions to acquire industrial skills and experience in their approved course of study.
2. To prepare students for the industrial works situation which they are likely to meet after graduation.
3. To expose students to work methods and techniques in handling equipment and machinery not available in their institutions.
4. To provide students with an opportunity to apply their knowledge in real work situation thereby bridging the gap between theory and practical.
5. To enlist and strengthen employers' involvement in the entire educational process and prepare students for employment in Industry and Commerce.

Chapter 2

*ip*NX Nigeria Limited

2.1 History and Corporate Profile

2.1.1 History

With its headquarters at 4, Balarabe Musa Crescent, Victoria Island, Lagos and several branch offices in Lagos, Ibadan, Abuja, Kano and Port Harcourt, *ip*NX Nigeria Limited is a leading provider of infrastructure-based telecommunication and information technological services based here in Nigeria. With more than a decade of experience, the company was formed by the divestment of the telecommunications services division of Telnet Nigeria Limited and has been in operations for over fifteen (15) years. *ip*NX Nigeria Limited started as a business division of Telnet Nigeria Limited - the leading indigenous Telecommunication and Information Technology Services Company in Nigeria. Telnet started business in 1987 as telecommunications engineering company and grew into other areas of information technology as technology evolved and opportunities arose. A major part of the business of telnet was providing data communication services, mainly wide area networks to Corporate communities in Nigeria in the Oil and Gas as well as Financial Services industries. Since NITEL was the monopoly provider of telecommunication services in Nigeria because of government regulation, these networks had to be built with NITEL facilities. In December 1992, the Federal Government of Nigeria deregulated the telecommunications industry, thereby opening it up to competition and other organisations could provide telecommunications services. Telnet saw this as an opportunity to improve its services to its customers and started to build its own communications network using radio and VSAT (satellite) technologies. The radio networks are utilised for communications between locations in the same metropolitan area while VSAT networks were mainly used to provide long-distance communications. The corporate organisations used the networks provided for both private voice and data communications.

In 2001 Telnet decided to separate the infrastructure-based business from the engineering (or knowledge) based business. This was done to:

- To allow Telnet Nigeria Limited provide engineering services to other infrastructure based service providers who might see Telnet as a competition.
- To allow other investors to invest in the infrastructure based business, which is very capital intensive.

The infrastructure-based Services Company within Telnet was therefore detached from the group to form a new company – "Netco Services Limited". Some Nigerian Communications Commission (NCC) licenses with Telnet were transferred to Netco and Netco got some additional licenses from the Nigerian Communications Commission (NCC). In total Netco has:

- Regional 3.5GHz FWA (Fixed Wireless Access) licenses in Lagos, Cross River, Bayelsa and Abuja.
- National VSAT license.
- Internet Service Provider License.

Unfortunately, a subsidiary of the Nigerian National Petroleum Company (NNPC); the National Engineering and Technical Company had also been known in the Nigerian environment with the acronym Netco and this caused a bit of confusion in the marketplace, hence in April 2003 the name of the company was changed to *ip*NX Nigeria Limited.

*ip*NX has now obtained additional frequencies from the NCC to be used to provide consumer and small businesses with Internet and Data Communications. These have been allocated to Lagos, Abuja (FCT), Cross Rivers and Bayelsa States with an opportunity to go to other states in Nigeria after we have started providing services in these locations.

2.1.2 Corporate Profile

*ip*NX is one of Nigeria's fastest growing Information and Communications Technology companies, serving a multitude of needs across enterprises, small businesses and residents with innovative, world-class services.

Our ability to identify, satisfy and exceed today's market needs is a testament to over a decade of experience, our commitment, drive and passion realised through highly skilled and well-seasoned professionals.

As a pioneer and a leading Fibre-To-The-Home (FTTH) operator in Nigeria, we currently provide several solutions to various industries and market segments using industry-leading technology (such as our very own Fibre-To-The-Home (FTTH) cable technology) as our core access network infrastructure and fixed wireless radio services (via licensed frequency).

We also proffer complementary IT solutions, with a view of covering key commercial and suburban regions.

Vision Statements of *ip*NX

To be the preferred communications and IT enabler in Africa and beyond.

Mission Statements of *ip*NX

Leveraging technology to create innovative solutions that help mankind thrive.

Core Credentials

Over 15 years experience

Trustworthy and devoted

More than 5,000 large and small business customers

Over 800km of cutting-edge fibre-optic cable infrastructure

*ip*NX is also the dependable service provider to the financial services sector, connecting all banks in Nigeria, Central Bank, InterSwitch & NIBSS.

Services

Broadband Internet

IP-VPN/MPLS

IP Wholesale

Telephony

Collocation & Hosting

Unified Communication

Cloud Computing

2.2 Why *ip*NX?

At *ip*NX, we pride ourselves on being game changers. We value smart, talented, hard-working people who stand tall and resolute amidst challenges. If great ideas thrill you, then you will fit in with us at *ip*NX. You will enjoy a stimulating environment, totally cool colleagues and all the opportunity your drive can handle.

2.2.1 Our Culture

Everyone in *ip*NX has the mindset of the explorer. We are always bold. We are to known venture into unknown and uncharted territories, yielding technology - enabled communications solutions that deliver optimal value to our stakeholders.

We constantly innovate and break boundaries. We are relentlessly customer obsessed, starting all we do with our customer in mind.

At *ip*NX, each employee takes personal responsibility and full accountability for upholding a culture that is inclusive, ethical and supportive of our corporate vision, goals and value.

2.2.2 Our Guiding Principles

We don't go out of our way to be different - We are. Our exciting and unique office culture is driven by who we are in the real world. From the word go, we focus on providing every person, every home and every business in Nigeria with world-class information, communication and entertainment services, while having fun.

Customer Obsession

At *ip*NX we obsess about our customers, and how to meet their requests. We pay attention to every detail that concerns them, with patience, true care, concern, constant follow up and follow through. We start and end all we do with our customers in mind.

Explorer

We are willing to try something new, venture into unchartered territories. Have a Sense of adventure, dare to be different. Be bold!

Can Do Spirit

Anything is possible! With hard work and strong commitment there is nothing we cannot achieve.

Teamwork

By working together, complementing each other's diversity, we achieve more.

Innovation

We constantly innovate in our processes, technology, products and services to deliver value.

Continuous Improvement

We continuously look to improve everything we do.

Bias for Action

Speed matters in business. We push relentlessly for desired outcomes and focus on results… the process is not the output! We value calculated risk taking.

Excellence

We are committed to relentlessly pursue excellence.

Continuous Learning

We will continuously improve our skills and knowledge in order to remain the best at what we do.

Respect for the Individual

Creating an open and enabling environment where different opinions can be expressed without fear. We value the contribution of everybody at every level. We treat our colleagues, whether supervisors, peers, or subordinates, with courtesy and respect, without harassment, or physical or verbal abuse. We acknowledge our rights to diversity yet are united in achieving our corporate goals.

Professionalism

At all times, we will exhibit strict compliance to the tenets of our profession and work environment.

Integrity

We deliver on promises to ourselves, clients and colleagues, holding ourselves to high levels of ethical behavior.

Ownership

Ownership is about taking responsibility and accepting accountability. I am accountable for the timeliness and quality of an outcome designated to me, even when working with others.

Frugality

We will achieve more with less, without compromising on quality. Our frugality inspires us to be resourceful, innovative and self-sufficient.

Exciting Work Environment

We believe our workplace should also be fun and creative, making it a place where people want to be every minute; where ideas thrive. This makes us more productive and keeps us engaged. We work smart …and we play smart.

2.2.3 Training And Development

We recognize at *ip*NX that learning improves and drives the company to its success. Our commitment to learning and development at *ip*NX is not just lip service but an actual dedicated effort that strengthens and empowers our employees throughout the duration of their career with us.

We offer amazing internal resources that include:

- ICademy online
- Employee development programmes
- Mentorship programmes
- Leadership development programmes

2.2.4 Compensation and Benefits

Work can be demanding but fun. However, at *ip*NX, we ensure that you stay rewarded for your performance and contribution. Compensation is determined by a number of

factors including company performance, divisional performance, and individual performance.

Our benefits generally include:

- Insurance
- Paid vacation time
- Paid paternity and maternity leave
- Other position specific benefits, amidst other options

2.2.5 Corporate Social Responsibility

Our aim is to support our communities by empowering them primarily through technology-enabled. Leveraging on technology, we contribute to the growth of the young population preparing them for independence and helping them to become useful members of society while achieving self-actualisation.

Through the use of technology, we provide educational services that close the gap between our children and children in a more developed economy.

Over the years, *ip*NX has improved the quality of education in the communities where she has done business.

2.3 *ip*NX Organogram

*ip*NX, being a big company, is divided into various divisions with divisional Chief Executive Officers (CEOs) all reporting to the Group Managing Director (GMD), Ejovi Aror. From the foregoing, *ip*NX currently has the organogram shown in Figure 2.1.

2.4 *ip*NX Departments And Divisions

*ip*NX provides a range of network connectivity, and delivery of internet, telephony, television as well as cloud-based software application services for our clients. These services are delivered by teams working across multiple divisions and departments.

2.4.1 *ip*NX Departments

Human Capital Management

Our Human Resource team brings out the best potential in every staff because here *ip*NX, our people matter. Their talent, commitment, energy – plus the million and one

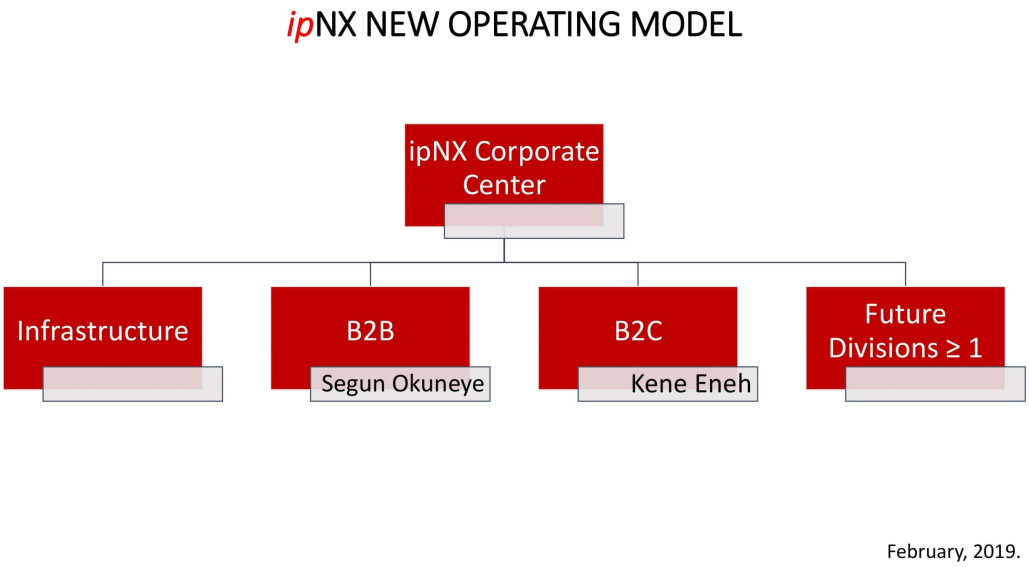
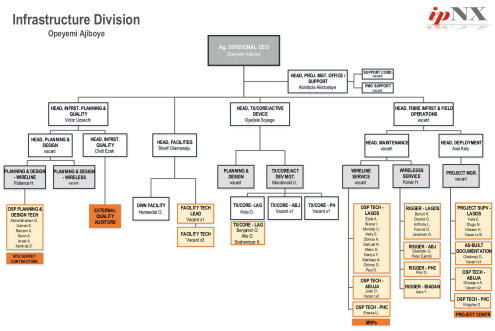
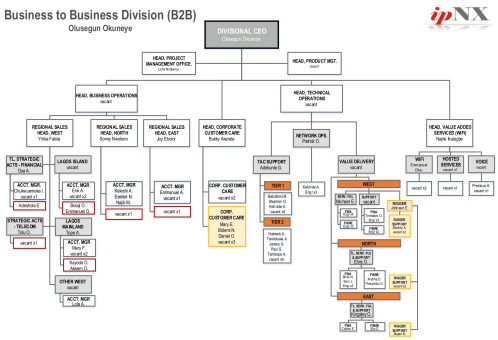


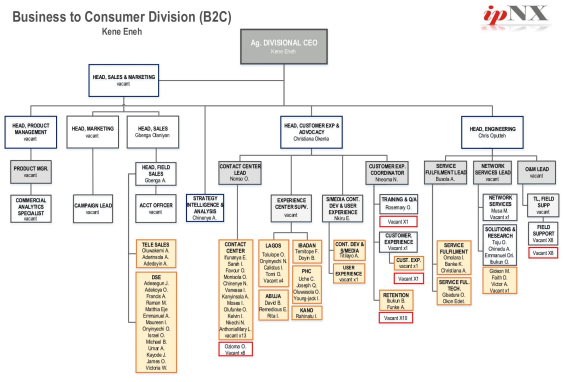
Figure 2.1: *ip*NX Main Organogram



(a) Infrastructure Organogram



(b) Business Division(B2B) Organogram



(c) Retail Division(B2C) Organogram

Figure 2.2: Each division's organogram

things they bring to our business- makes us who we are. The way you think matters to us. we value creative thinking. So far, it has helped us recreate rules and we want more of it. If you like adventure, if you don't give up, if you are focus on getting results, we want you at *ip*NX.

Finance

We would be nowhere without the expertise and insight of our finance teams. Our various teams include: Financial Accounting Revenue Assurance Treasury Billing The teams are there to understand, support and challenge the business, making sure we are doing everything we can to run efficiently by providing the financial expertise required to grow our business to profitable levels and heights. If you have an eye for numbers and a brain for planning, what are you waiting for?

Legal, Risk And Compliance

Our Legal area has a big impact on our actions. Fortunately, we can rely on the counsel, creative thinking and sound judgment of the Law, Risk and Compliance team to unravel the finer points and keep us on the right track. If you know you are a champion of business integrity; are good at building constructive business relationships with our customers, partners, local communities, or the government; strive to protect and promote innovation, enterprise, then join us for a career with our LRC team.

Customer Experience & Advocacy

When our customers need support and advice, or they want to get something sorted out, they deserve the right help, right on time. Everyone in customer service loves to make a difference because they strive for excellence. They won't stop until they can see the smile of satisfaction on the customer's face. Very simply put, we obsess over our customer. On joining the Customer service team, you will be the voice of *ip*NX. And whatever the next call brings, your goal remains the same; to delight and dazzle with your knowledge and commitment. If you can throw in some pleasant surprises along the way, everyone is a winner.

Health, Safety And Environment

*ip*NX Health, Safety and Environmental team is responsible for development, oversight, and management of environmental health and safety programs that protect the *ip*NX working environment, provide safe and healthy conditions for work and comply with applicable laws and regulations. It is the team's goal is to pro vide responsive service and

critical support to ensure that *ip*NX is a safe and healthy environment in which to work, study, and live. If you are an expert in what you do and love safety, apply to join our HSE team.

Marketing

*ip*NX Marketing Department plays a massive role in promoting the mission and business of *ip*NX. *ip*NX Marketing team's main focus is to understand our customers, their values, and perception and make sure they get it – through our amazing services because *ip*NX was born to disrupt – but with a purpose. If you can wow our customers with *ip*NX brand, you might just fit right in with this team.

Internal Audit

Our Internal Audit team provides *ip*NX with unbiased, objective and constructive views that help companies succeed. Our internal audit team examines and evaluates the policies, procedures and systems which are in place to ensure the reliability and integrity of information, compliance with policies, plans, laws and regulations; the safeguarding of the assets; and the efficient use of resources. We are looking for professionals who take pride in their work.

Corporate Services

Our Corporate services are made up of Human Capital Management, Administrative Services, Facility Management, etc. They provide services to our people. They make sure we get the best employees who remain the best employees on top of their game; they make sure we have both the physically and mentally enabling environment to support and push out our creativity. At *ip*NX, we want to work with people who know the next best idea is just around the corner – and who have the ambition, creativity, and passion for going after it.

Supply Chain Management

Our SCM team helps cordinate and integrate seamless information, material and finance flow from suppliers to customers, so that we have an excellent customer service, an ideal inventory management and low unit cost. At *ip*NX, our SCM team has the prime responsibility of integrated planning, sourcing and deployment of material connected to our network services . If you know you have the expertise for this job, why not join our SCM team.

Information Systems And Technology

As an organization that is in love with leading edge innovation and services, it is no wonder that our IS&T team plays a crucial role. The team provides exceptional technological services and support. Our team enables efficiency, productivity and simplicity both internally and externally, helping us to grow. Joining our IS&T team means you will help keep our internal IT running smoothly by creating and maintaining our systems and applications.

Fibre Project And Rollout

Our Fibre Project and Rollout team are our hunchos when it comes to rolling out our network fibres direct to business areas and homes. If you have a passion for success and want to make a difference for customers every day, join our Fibre Project and Rollout Team.

Admin Services

Every organization needs somebody to coordinate it. That job is for the *ip*NX administrative team. The administrative team brings its conscientious planning, attention to detail and flexible thinking to the heap of projects. They help make sure things run like a well-oiled clock. This means organizing everything from staying on top of day to day administration, prioritizing and managing real estate and other assets, getting involved in team strategy and company – wide objectives. If you want to join our administration team, there is always plenty of opportunity to make your mark.

Research And System Architecture

The *ip*NX Research and System Architecture team is reinventing the technology world every day. The Research and System Architecture team is in charge of all our research and development and are also responsible for coming up with tomorrow services that meet the needs for our today's customers. They look for ways to do it cheaper, faster, and better. We are looking for those who have the drive and passion for technology.

2.4.2 *ip*NX Divisions

Infrastructure Division

*ip*NX network is always on duty, always in demand. That means it has to be reliable. This all comes down to the diligence and support of our network services team. Whether

fixed-voice and data; mobility; and IT services, they make sure our network is ever available. This means our customers' experience is in their hands. They are there to trouble shoot malfunctions and make the network service bigger, better, faster and stronger. If you are someone who can resolve issues, develop ideas to enhance a growing network and most importantly, work with the *ip*NX team and embrace the values we uphold, then you have what it takes to be part of *ip*NX community.

Retail Division

Business Division

Chapter 3

Projects Done and Experiences Gained

3.1 Preliminaries

This section gives a brief introduction to or exposes the technologies used while implementing the projects assigned to me.

3.1.1 Software development process overview at *ip*NX

In IEEE standard Glossary of Software Engineering Terminology, the SDLC is: "The period of time that starts when a software product is conceived and ends when the product is no longer available for use". This circle typically includes the following stages:

1. Planning and requirement analysis: Each SDLC model starts with Planning and requirement analysis, in which stakeholders of the process plan and discuss the requirements for the final product with the goal of a detailed definition of the system requirements.

2. Designing project architecture: In this stage, the developers engage in designing the architecture of the software product and any surfaced technical questions are discussed by all the stakeholders, including the customer. Technologies which will be used, team load, limitations, duration and budget are also defined.

3. Development and programming: Having approved the requirements stated, the process comes to this stage – actual development. Programmers incept writing of source codes to align with the defined requirements, System administrators adjust the software environment, front-end programmers develop the user interface of the program and the logics for its interaction with the server. The programming by itself assumes four stages:

- Development of Algorithms
- Writing of Source Codes
- Source codes compilation
- Unit testing and debugging

4. Testing: This is the phase where comprehensive testing and debugging of the developed software product take place. All the code flaws missed during the development are detected, documented, and passed back to the developers to fix. The testing process repeats until all the critical issues are removed and software workflow is stable.
5. Deployment: When the program is finalised and has no critical issues, it is time to launch it for the end users and this happens in this stage.
6. Maintenance

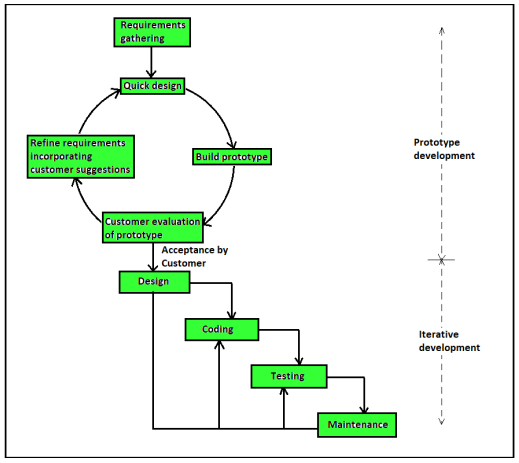
According to Sharma and Singh (2015), General software process models are:

1. Waterfall model
2. Prototype model
3. Rapid application development model (RAD)
4. Incremental model
5. Spiral model
6. Agile
7. V-shaped model

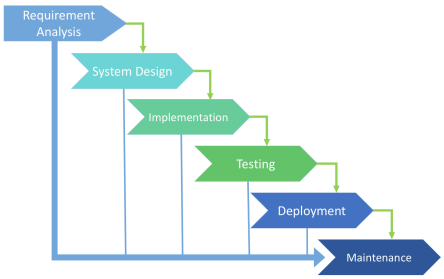
These models are depicted in Figure 3.1 below:

Comparison of SDLC Models

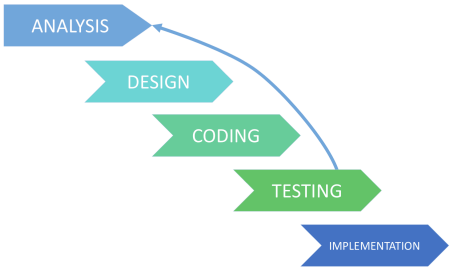
Table: 3.1 compares the General software process models on different Parameters as listed and compared in Sharma and Singh (2015).



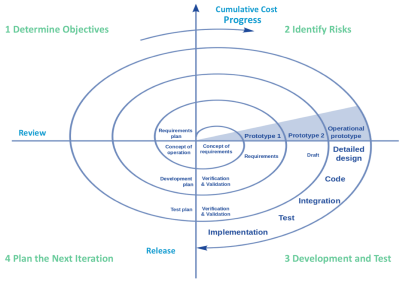
(a) Prototype Model, Source: Pal (2020)



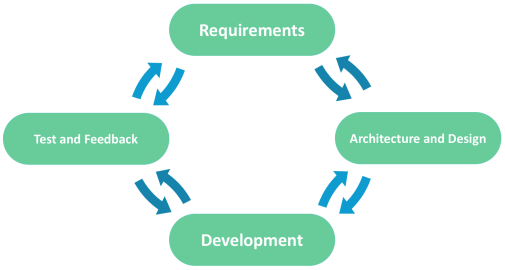
(b) Waterfall model, Source: Team (2020)



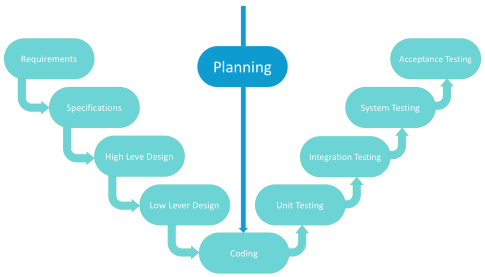
(c) Incremental model, Source: Team (2020)



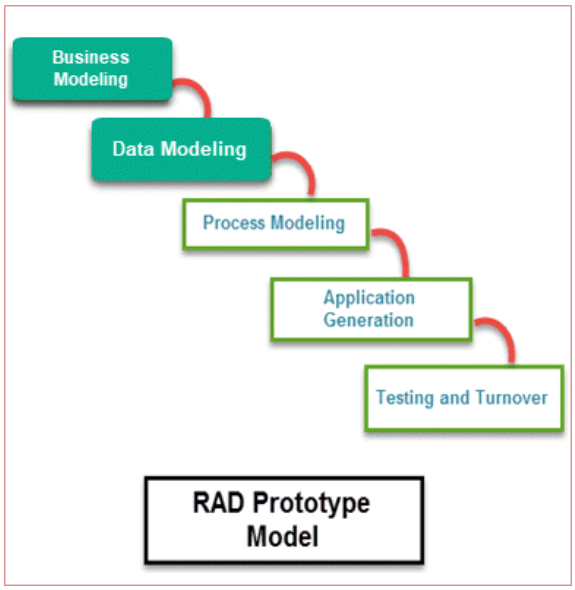
(d) Spiral model, Source: Team (2020)



(e) Agile model, Source: Team (2020)



(f) V-shaped model, Source: Team (2020)



(g) Rapid application development model (RAD)

Figure 3.1: SDLC Models

Table 3.1: Comparison of various SDLC Models on different Parameters

Model/Features	Waterfall	Prototype	RAD	Incremental	Spiral	Build & Fix	V-shaped
Well defined requirements	Yes	No	Yes	No	No	No	Yes
User involvement in all phases	Only at beginning	High	Only at beginning	Yes(Intermediate)	High	No	No
Risk analysis	Only at beginning	No Risk analysis	Low	No Risk analysis	Yes	No	Only at beginning
Overlapping phases	No overlapping	Yes	No	No	Yes	Yes	No
Implementation time	Long	Quick	Quick	Long	Long	Depend upon Project	Long
Cost	Low	High	Low	Low	Expensive	Low	Expensive
Incorporation of changes	Difficult	Easy	Easy	Easy	Easy	Difficult	Difficult
Simplicity	Simple	Simple	Simple	Intermediate	Intermediate	Simple	Intermediate
Flexibility	Rigid	Little flexible	High	Less flexible	Flexible	FLexible	Less flexible

3.1.2 Python

Python is the programming language in which Django and Flask frameworks, used in implementing majority of the projects, were written. It is a general-purpose, versatile and modern programming language with a dynamic scripting ability similar to Perl and Ruby. First released in 1991 by its principal author, Guido van Rossum, Python supports dynamic typing and has a garbage collector for automatic memory management. Another important feature of Python is the dynamic name solution which binds the names of functions and variables during execution, Zhou (2010).

3.1.3 Flask Framework

Flask is a lightweight Web Server Gateway Interface (WSGI) web application framework (Ronacher (2020)) written in Python by Armin Ronacher and first released on the 1st of April 2010. It was designed as an extensible framework from the ground up providing a solid core with the basic services while allowing extensions to provide the rest. Since one can pick and choose the required extension packages, a lean stack with no bloat and which does exactly what one needs is ended up with.

Flask has two main dependencies, namely Werkzeug and Jinja2. Werkzeug provides the routing, debugging, and WSGI subsystems, while template support is provided by Jinja2 Grinberg (2014).

3.1.4 Django Framework

Django is a powerful Python web application framework that encourages rapid development with clean, and pragmatic design, which offers a relatively shallow learning curve, Melé (2018). It is open source with the primary goal of making the development of complex and data-based websites easier. Thus it emphasizes the re-usability and ”plug-gability” of components to ensure rapid developments, Zhou (2010). Django consists of three layers, namely model, view and template, Holovaty and Willison (2019).

Model Layer

Model, the single and definitive source of information about a data, contains the essential fields and behaviours of the data being stored. Generally, each model maps to a single database table, Holovaty and Willison (2019).

View Layer

Django's "views", which can be function- or class-based, encapsulate the logic responsible for processing a user's request and for returning the desired response, Holovaty and Willison (2019). Response may be a HyperText Markup Language (HTML) content, XML document or error code such as error 404, 505 and so on. The logic encapsulated in a view may be arbitrary provided that the desired response is returned.

Template Layer

The template layer provides a designer-friendly syntax for rendering the information to be presented to the user. A template contains the static parts of the desired HTML output and some special syntax describing how dynamic content will be inserted and a Django project can be configured with one or several template engines or systems such as the Django template language (DTL) and Jinja2.

3.1.5 AJAX

AJAX, a term coined in 2005 by Jesse James Garrett, stands for Asynchronous JavaScript and XML and refers to a set of web development techniques combining existing web technologies, including HTML or eXtensible HyperText Markup Language (XHTML), Cascading Style Sheets, JavaScript, The Document Object Model, XML, eXtensible Stylesheet Language Transformations (XSLT), and most importantly the XHR object, Yang (2020). Ajax is widely used in client-side programming (e.g. JavaScript) to allow for data to be sent and received to and from a database or server without actually disturbing the user experience or reloading the entire browser page. Figure 3.2 compares the conventional method of requesting data from a web server and the Ajax method.

AJAX was greatly utilised in carrying out my projects since most of them exhibited Real-time functionality. Code Snippet 3.1 is a sample snippet from one of the projects with asynchronous data persistence.

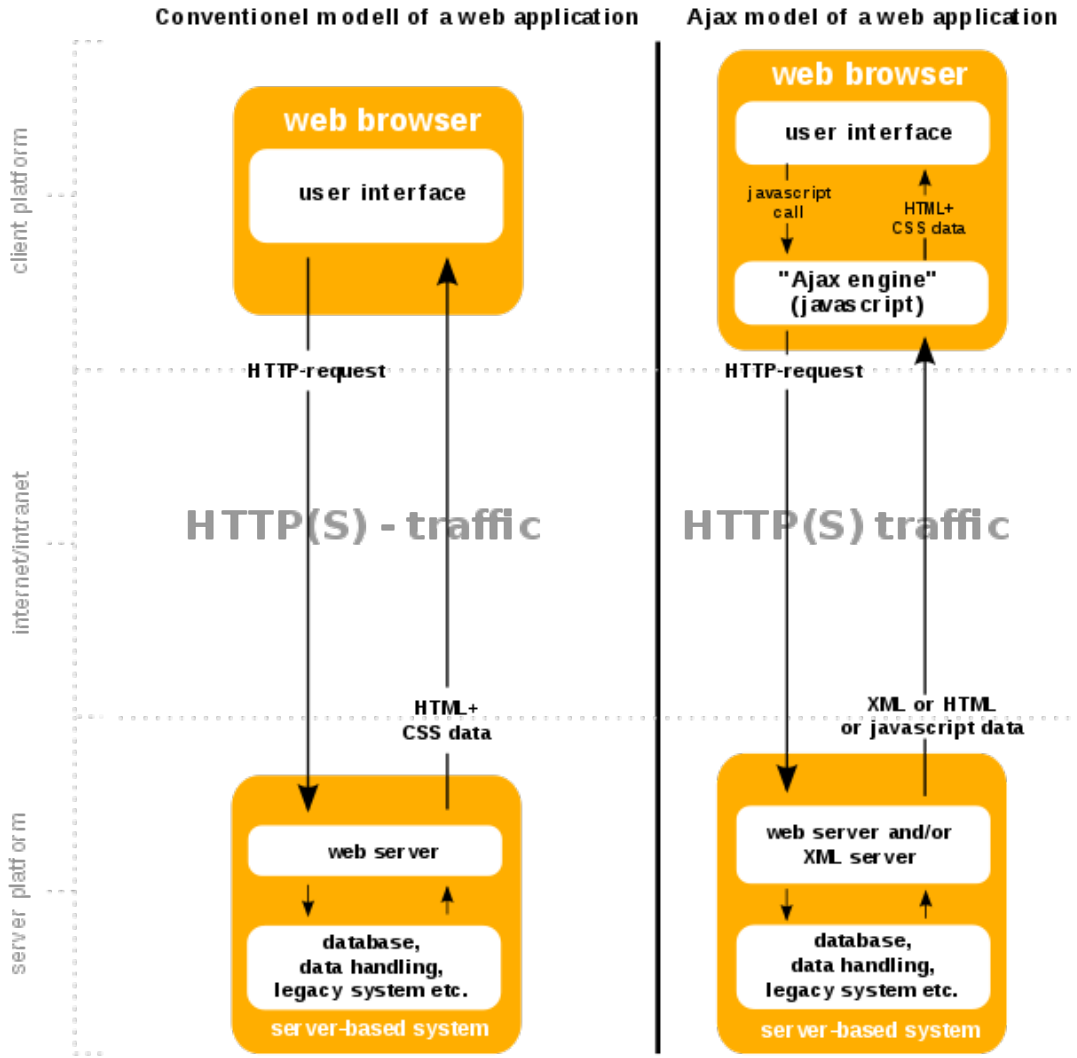


Figure 3.2: Conventional method vs AJAX method, Source Wikipedia.com

Code Snippet 3.1: Django Blog's AJAX snippet

```
$(document).ready(function(event) {
        $(document).on('submit', '#postcomment', function(event) {
        event.preventDefault();
        $.ajax({
                type: 'POST',
                url: $(this).attr('action'),
                data: $(this).serialize(),
                dataType: 'json',
                success: function(response) {
                        $('.comments-wrap').html(response['form']);
                                $(document).ready(function() {
                                        $("#commenters").on("click", ".reply",
                                        function(event) {
                                                event.preventDefault();
                                                var form =
                                                ↪  $("#postcomment").clone(true);
                                                form.find('.parent').val($(this)
                                                        .parent().parent().attr('id'));
                                                $(this).parent().append(form).fade();
                                });
                        });
                        $(document).ready(function() {
                                var ssPrettyPrint = function() {
                                        $('pre').addClass('prettyprint');
                                        $('code').addClass('prettyprint');
                                        $(document).ready(function() {
                                                PR.prettyPrint();
                                        });
                                };
                        });
                },
                error: function(rs, e) {
                        console.log(rs.responseText);
                },
        })
});
});
```

The above snippet was the underlying code responsible for the real-time functionality of the Django blog's commenting system. It updates and fetches data from the Comment table which had been created in the blog's models without reloading the entire web page.

3.1.6 SQLAlchemy

According to Bayer (2016), The SQLAlchemy Structured Query Language (SQL) Toolkit and Object Relational Mapper (ORM) provide a comprehensive set of tools for working with databases and Python. It consists of several distinct areas of functionality which can be individually or collaboratively used together. Below is the illustration of the

major components of SQLAlchemy, with component dependencies organized into layers:

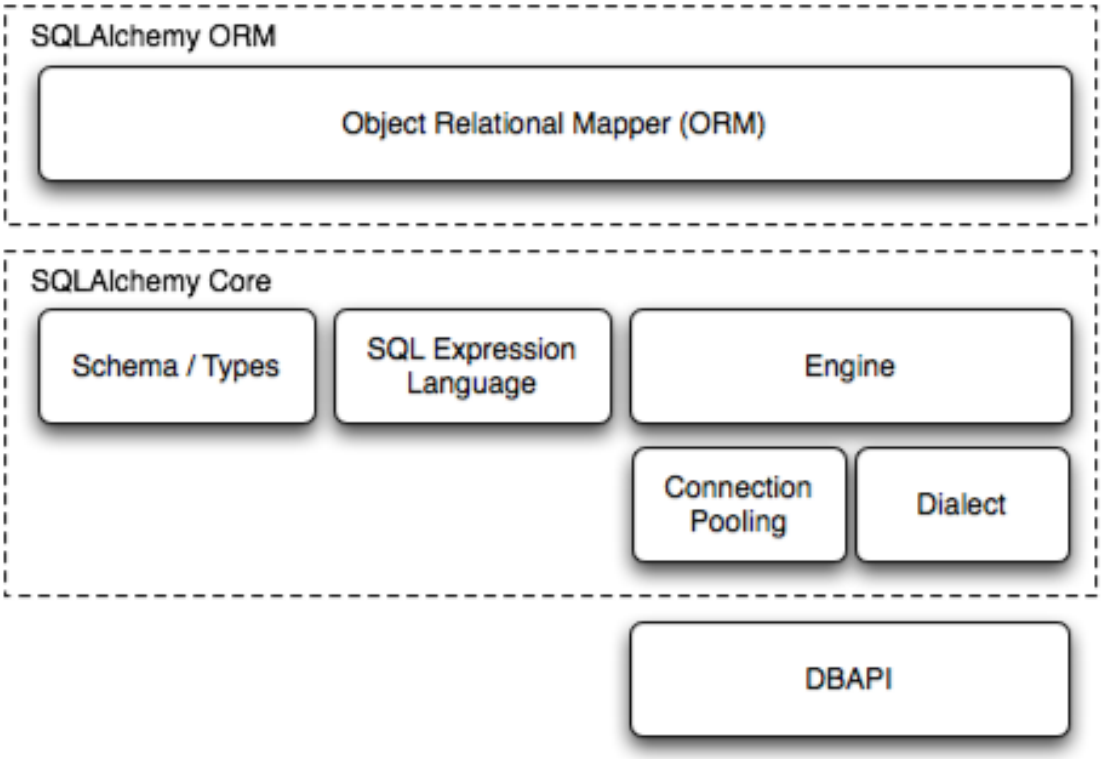


Figure 3.3: SQLAlchemy overview, from Bayer (2016)

From the above figure, the two most significant front-facing portions of SQLAlchemy are the ORM and the SQL Expression Language. SQL Expressions can be used independently of the ORM. When using the ORM, the SQL Expression language remains part of the public facing Application Programming Interface (API) as it is used within object-relational configurations and queries, Bayer (2016).

3.1.7 Jinja2

Jinja2 is a fast, expressive, extensible and modern day web templating language or engine for Python developers mimicking DTL with the inclusion of call functions with arguments on objects. From Ronacher (2017), An important usage of Jinga2 is the creation of HTML, XML or other markup formats which are returned to the user via HyperText Transfer Protocol (HTTP) response.

3.2 Python-based Projects

3.2.1 Methodology

All the web applications were built using Flask or Django framework at the Back-end and Creative Tim's dashboard and other extensible templates at the front-end following the procedures below:

Created the Database tables

To get started, a back-end of the system which is the database was created. It should be noted that implementing this system was framework dependent as discussed below.

- **Flask**: One of the notable tables in the database was the User table which was created by writing a Class that inherited from Flask's model class. A snippet is:

Code Snippet 3.2: vCPE Flask User Model

```
from app          import db
from flask_login import UserMixin

from . common     import COMMON, STATUS, DATATYPE

class User(UserMixin, db.Model):

        id          = db.Column(db.Integer,     primary_key=True)
        user        = db.Column(db.String(64),  unique = True)
        email       = db.Column(db.String(120), unique = True)
        name        = db.Column(db.String(500))
        role        = db.Column(db.Integer)
        password    = db.Column(db.String(500))
        password_q  = db.Column(db.Integer)

        def __init__(self, user, password, name, email):
                self.user       = user
                self.password   = password
                self.password_q = DATATYPE.CRYPTED
                self.name       = name
                self.email      = email
                self.group_id = None
                self.role     = None

        def __repr__(self):
                return '<User %r>' % (self.id)

        def save(self):

                # inject self into db session
                db.session.add ( self )

                # commit change and save the object
```

```
34                 db.session.commit( )
35
36                 return self
```

This piece of code creates a table called `User` which stores the user information. The table has seven columns: `id`, a primary key; `user` which corresponds to `User`'s username; `email` corresponding to `User`'s email; `name` corresponding to the name of the `User`; role; `password` and `password_q` corresponding to the password and confirm password fields of the `User`. There are also the `__init__`(Python's Object constructor), `__repr__`, and `save` methods all taking *`"self"`* as an argument, a requirement in Python programming language for methods. `__init__` and `__repr__` are called *dunder* or Double Under(Underscores) methods which are sometimes referred to as Magic methods. When an object of the class `User` is created, the `__init__` method is invoked while `__repr__` and its counterpart, `__str__`, found in Code Snippet 3.2 line 31, controls the `to-string` conversion in the class that houses it.

Flask uses `SQLAlchemy` for the ORM or Data Access part. It takes charge of the database management such as creating tables, writing SQL queries and other database operations.

- **Django**: For Django, creation of tables is fairly more verbose but simple. The snippet below buttresses this point.

Code Snippet 3.3: Django Blog's PublishedManager, Category,and Post

```
1  from django.contrib.postgres.fields import ArrayField
2  from django.db import models
3  from django.contrib.contenttypes.models import ContentType
4  from django.contrib.auth.models import User
5  from ckeditor_uploader.fields import RichTextUploadingField
6  from ckeditor.fields import RichTextField
7  from django.utils import timezone
8  from django.urls import reverse
9  from django.db.models.signals import pre_save
10 from PIL import Image
11 from taggit.managers import TaggableManager
12 from .utils import get_read_time
13 from django.utils.safestring import mark_safe
14 #from django.contrib.contenttypes.fields import GenericRelation
15
16 # Create your models here.
17
18 class PublishedManager(models.Manager):
19         def get_queryset(self):
20                 return super(PublishedManager, self).get_queryset()
21                 .filter(status='published')
```

```
22
23  class Category(models.Model):
24          name = models.TextField(max_length=1000000)
25          slug = models.SlugField(max_length=10000000, unique=True)
26
27          class Meta:
28                  ordering = ('name',)
29                  verbose_name='category'
30                  verbose_name_plural = 'categories'
31
32          def __str__(self):
33                  return self.name
34
35  class Post(models.Model):
36          STATUS_CHOICES = (
37                  ('draft', 'Draft'),
38                  ('published', 'Published'),
39          )
40          title   = models.CharField(max_length=10000000)
41          slug    = models.SlugField(max_length=10000000,
            ↪  unique_for_date='publish')
42          author  = models.ForeignKey(User, on_delete=models.CASCADE,
43          related_name='blog_posts')
44          image   = models.ImageField(upload_to='posts_pics'
45          ,default='logo.svg')
46          body    = RichTextUploadingField()
47          publish = models.DateTimeField(default=timezone.now)
48          read_time =  models.IntegerField(default=0)
49          created = models.DateTimeField(auto_now_add=True)
50          updated = models.DateTimeField(auto_now=True)
51          status  = models.CharField(max_length=100
52          ,choices=STATUS_CHOICES, default='draft')
53          objects = models.Manager() # The default manager.
54          published = PublishedManager() # Our custom manager.
55          category = models.ForeignKey(Category, on_delete=models.CASCADE)
56          tags = TaggableManager()
57          class Meta:
58                  ordering = ('-created',)
59
60          def __str__(self):
61                  return self.title
62
63
64          def get_absolute_url(self):
65                  return reverse('blog:post_detail',args=[self.publish.year
66                  , self.publish.month, self.publish.day, self.slug])
67
68          def get_next_post(self):
69                  return self.get_next_by_created()
70
71          def get_previous_post(self):
72                  return self.get_previous_by_created()
73
74          def get_markdown(self):
75                  body = self.body
76                  return mark_safe(body)
```

```
77
78          @property
79          def get_content_type(self):
80                  instance = self
81                  content_type =
                    ↪  ContentType.objects.get_for_model(instance.__class__)
82                  return content_type
83
84          def save(self, *args, **kwargs):
85                  super(Post, self).save(*args, **kwargs)
86
87                  img = Image.open(self.image.path)
88                  width = 883
89                  height = 391
90                  if img.width > width or img.height > height:
91                          output_size = (width, height)
92                          img.thumbnail(output_size)
93                          img.save(self.image.path)
```

Created the User Interface

Although the Database, where tables of any kind could be updated using Django and/or Flask models, had been created, it would be a disaster for users and the technical maintenance team if end users are allowed to manipulate the data directly. Therefore, nice and responsive user interfaces were created to let users interact with the data indirectly. As discussed above, Flask and Django provide templating components to create the user interfaces. Since Flask solely uses Jinja2 whereas Django has an optional DTL, the syntax for creating the user interfaces are a bit different.

- **Django**: Django supports two templating engines, namely Jinja2 and DTL where the latter happens to be the default with the below snippet.

Code Snippet 3.4: Django Blog's `post_form.html`

```
1  {% extends 'base.html' %}
2  {% block header %}{% endblock header %}
3  {% block content %}
4  <!-- site content
5  ============================================== -->
6  <div class="s-content content">
7          <main class="row content__page">
8                  <section class="column large-full entry format-standard">
9                          <div class="content__page-header">
10                                 <h1 class="display-1">
11                                         Write A Post.
12                                 </h1>
13                         </div> <!-- end content__page-header -->
14                         <form name="contactForm" id="contactForm"
                           ↪  method="post"
```

```
15                          action="" autocomplete="on"
                            ↪  enctype="multipart/form-data">
16                                  {% csrf_token %}
17                                  <fieldset>
18                                          {% for field in form %}
19                                                  <div class="form-field">
20                                                          {{ form.media }}
21                                                          {{ field.label }}
22                                                          {{ field }}
23                                                  </div>
24                                          {% endfor %}
25                                          <input name="submit" id="submit"
                                            ↪  class="btn
26                                          btn--small" value="Create Post"
    ↪  type="submit">
27                                  </fieldset>
28                          </form> <!-- end form -->
29                  </section>
30          </main>
31  </div> <!-- end s-content -->
32  {% endblock content %}
```

Every line surrounded by {% %} is a Django sentence which deals with simple *{% if-else %}*, *{% csrf_token %}* and *{% for-loop %}* among others. The lines surrounded by double curly braces, such as *{{ field }}*, denote variables whose values are passed from view functions whenever this template has to be displayed.

- **Flask**: For flask, Jinja2 is one of the major dependencies needed to get it up and running and the syntax looks as follows:

Code Snippet 3.5: CEA Dashboard `login.html`

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1">
6   {% if title %}
7           <title>ipNX dashboard - {{ title }}</title>
8   {% else %}
9           <title>ipNX dashboard</title>
10  {% endif %}
11  </head>
12  <body>
13          <div class="limiter">
14                  <div class="container-login100">
15                          <div class="wrap-login100">
16                                  <form class="login100-form validate-form"
17                                  method="POST" action="">
18                                          {{ form.hidden_tag() }}
```

```
19                                                  <span class="login100-form-logo">
20                                                      <img
21                                                          src="/static/assets/custom/login
22                                                          /images/logo.png"
    ↪  alt="">
23                                                  </span>
24                                                  <span class="login100-form-title
                                                    ↪  p-b-34 p-t-27">
25                                                      Log in
26                                                  </span>
27                                                  {% for message in
                                                    ↪  get_flashed_messages() %}
28                                                      <div class="text-center
                                                        ↪  p-t-5 p-b-5">
29                                                          <p class="error">
30                                                              {{ message
                                                                ↪  }}
31                                                          </p>
32                                                      </div>
33                                                  {% endfor %}
34                                                  <div class="wrap-input100
                                                    ↪  validate-input"
35                                                  data-validate="Enter E-mail">
36                                                      {{
                                                        ↪  form.email(class="input100",
37                                                          placeholder="E-mail")
                                                            ↪  }}
38                                                      <span
                                                        ↪  class="focus-input100"
39                                                          data-placeholder="&#xf207;">
40                                                      </span>
41                                                  </div>
42
43                                                  <div class="wrap-input100
                                                    ↪  validate-input"
44                                                  data-validate="Enter password">
45                                                      {{
                                                        ↪  form.password(placeholder="Password",
46                                                          class="input100")
                                                            ↪  }}
47                                                          <span
                                                            ↪  class="focus-input100"
48                                                              data-placeholder="&#xf191;">
49                                                          </span>
50                                                  </div>
51
52                                                  <div
                                                    ↪  class="contact100-form-checkbox">
53                                                      {{
                                                        ↪  form.remember(class="input-checkbox100",
54                                                          id="ckb1",type="checkbox")
                                                            ↪  }}
55                                                      <label
                                                        ↪  class="label-checkbox100"
                                                        ↪  for="ckb1">
56                                                          Remember me
```

```
                                        </label>
                                    </div>

                                    <div
                                    ↪  class="container-login100-form-btn">

                                        {{
                                        ↪  form.submit(class="login100-form-btn")
                                        ↪  }}

                                    </div>

                                    <div class="text-center p-t-10">
                                        <a class="txt1" href="#">
                                            Forgot Password?
                                        </a>
                                    </div>
                                </form>
                            </div>
                        </div>
                    </div>

</body>

</html>
```

3.2.2 Implemented the view function

Having dealt with the back-end databases and the frontend web pages user interfaces, the logic which stands in between to deal with the user requests and maintain the database were implemented. Unsurprisingly, the view systems in Flask and Django are different.

- **Django**: Django's views component provides a set of API to implement the logic. The Django view file, `views.py`, houses the functions and/or classes to achieve all set goals. The view functions and/or classes are shown in the snippet below:

Code Snippet 3.6: Django's Blog `views.py`

```
from django.shortcuts import render, get_object_or_404, redirect
from django.contrib.contenttypes.models import ContentType
from django.template.defaultfilters import slugify
from django.http import HttpResponse, HttpResponseRedirect,
Http404, JsonResponse
from django.core.paginator import Paginator,
EmptyPage, PageNotAnInteger
from django.views import generic
from django.contrib.auth.mixins import LoginRequiredMixin
, UserPassesTestMixin

from taggit.models import Tag
from django.contrib.auth.decorators import login_required
```

```
14 from django.db.models import Count, Q
15 from django.contrib.auth.models import User
16 from .models import Post, Comment
17 from .forms import PostForm, UpdatePostForm, CommentForm
18 from django.template.loader import render_to_string
19 from django.contrib.postgres.search import SearchVector,
20 SearchQuery, SearchRank, TrigramSimilarity
21
22 #Function based view
23 def blog_index(request, tag_slug=None):
24         object_list = Post.published.all()
25         if request.user.is_staff or request.user.is_superuser:
26                 object_list = Post.objects.all()
27         common_tags = Post.tags.most_common()[:2]
28         tag = None
29
30         if tag_slug:
31                 tag = get_object_or_404(Tag, slug=tag_slug)
32                 object_list = object_list.filter(tags__in=[tag])
33
34         paginator = Paginator(object_list, 5) # 5 posts in each page
35         page = request.GET.get('page')
36         try:
37                 posts = paginator.page(page)
38                 except PageNotAnInteger:
39                 # If page is not an integer deliver the first page
40                 posts = paginator.page(1)
41         except EmptyPage:
42                 # If page is out of range deliver last page of results
43                 posts = paginator.page(paginator.num_pages)
44         context = {
45                 'page': page,
46                 'posts': posts,
47                 'tag': tag,
48                 'common_tags': common_tags,
49         }
50         return render(request, 'blog/index.html', context)
51
52 #Class based view
53 class PostUpdateView(LoginRequiredMixin,
54                 UserPassesTestMixin, generic.UpdateView):
55         model = Post
56         form_class=UpdatePostForm
57         template_name = 'blog/post_form.html'
58
59         def form_valid(self, form):
60                 form.instance.author = self.request.user
61                 return super().form_valid(form)
62
63         def test_func(self):
64                 post = self.get_object()
65         if self.request.user == post.author or self.request.user.is_staff:
66                 return True
67         return False
```

- **Flask**: According to Grinberg (2014), "Clients such as web browsers send requests to the web server, which in turn sends them to the Flask application instance. The application instance needs to know what code needs to run for each URL requested, so it keeps a mapping of URLs to Python functions. The association between a URL and the function that handles it is called a `route`. The most convenient way to define a route in a Flask application is through the `app.route` decorator exposed by the application instance, which registers the decorated function as a `route`. The following example shows how a route is declared using this decorator:

Code Snippet 3.7: Example snippet

```
@app.route('/')
def index():
        return '<h1>Hello World!</h1>'
```

Functions like index() are called view functions" and are mostly housed in the `views.py` file and an example of a comprehensive `views.py` file is shown below:

Code Snippet 3.8: An excerpt from CEA dashboard's `views.py` file

```
# all the imports necessary

from flask import json, url_for, redirect, Markup,
Response, render_template, flash, g, session, jsonify,
request, send_from_directory
from werkzeug.exceptions import HTTPException, NotFound, abort

import os
import secrets
from PIL import Image
from app  import app

from flask       import url_for, redirect, render_template,
flash, g, session, jsonify, request, send_from_directory
from flask_login import login_user, logout_user,
current_user, login_required
from app         import app, lm, db, bc
from . models    import User
from . common    import COMMON, STATUS
from . assets    import *
from . forms     import LoginForm, RegisterForm, UpdateAccountForm
import os, shutil, re, cgi, json, random, time
from datetime import datetime


random.seed()  # Initialize the random number generator

# provide login manager with load_user callback
@lm.user_loader
def load_user(user_id):
```

```
31          return User.query.get(int(user_id))
32  @app.route('/index')
33  @login_required
34  def index():
35          inactiveCustomers = 34546
36          activeCustomers = 7654984
37          numberOfCalls = 123456
38          numberOfRetailCustomers = 455674666
39          return render_template('pages/index.html',
40          numberOfCalls=numberOfCalls,
41          numberOfRetailCustomers=numberOfRetailCustomers,
42          inactiveCustomers=inactiveCustomers,
43          activeCustomers=activeCustomers)
44
45
46  # authenticate user
47  @app.route('/logout')
48  def logout():
49          logout_user()
50          return redirect(url_for('login'))
```

3.2.3 Projects Implemented

Some of the projects implemented using the Python programming language and its two most popular frameworks, Django and Flask, are undermentioned and discussed.

Customer Premises Equipment (CPE)

- **Project's Requirement specification**: The project was to "build a web UI for a customer premises equipment (CPE) through which settings can be modified.

  Please design such a web UI with the following pages:

  * Login page - username and password
  * Main page - device summary showing WAN and LAN IPv4 addresses
  * Settings page -

    Enable/disable DHCP service, modify address range, manage DHCP reservations

    Manage DNS servers

    Manage device whitelist/blacklist based on MAC addresses

    Manage WiFi settings including ESSID name and pre-shared key (password), enable/disable 2.4GHz and/or 5GHz bands, add or remove WiFi devices

This must be a responsive site with a nice UI and interaction. Communication between front-end and back-end should JSON over XHR asynchronously. Backend should be Python or PHP based with data persistence in SQLite on the backend server".

Having implemented the features stated using Flask, the system was initiated and Figure 3.3 shows the pages of each feature:

Customer Experience Analysis (CEA) Dashboard

- **Project's Requirement specification**: Customer Experience Analysis (CEA) Dashboard's requirement specification was to develop or implement a real-time dashboard which has the metrics and features as shown in the Figure 3.5 below:

Figure 3.6 depicts the looks of the software product which was wholly implemented in Python's Flask:

Data Usage Analytic (DUA) Dashboard

- **Project's requirement specification**: *ip*NX already had a Data Usage Analytic (DUA) Dashboard but was with poor design and user interface. The task was to re-engineer, re-develop and revamp the whole system with the following additional features:

  Animated Error page for advertisement.

  Modal views of preliminary Data Usage analysis of each device on each user's network.

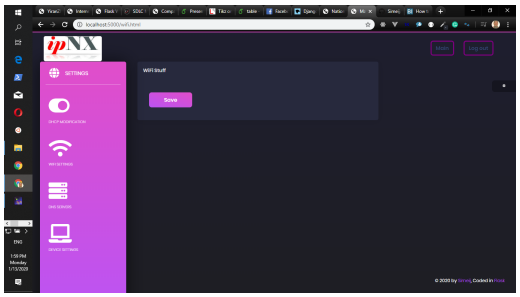
  Attractive Login page with company's products advertisement as carousels.

  Responsive Dashboard theme selection pluggin.

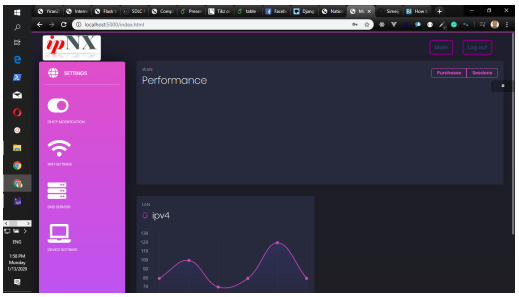
Figures 3.7 and 3.8 show the initial system as well as its re-designed, using Flask micro-framework, counterpart.

Django Blog

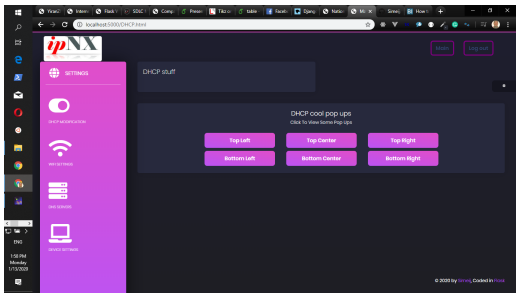
- **Project's description**: **Devc**, the name of the blog, is a complex web application inspired by **Frances Nnamadim**, a senior and regular colleague who worked as



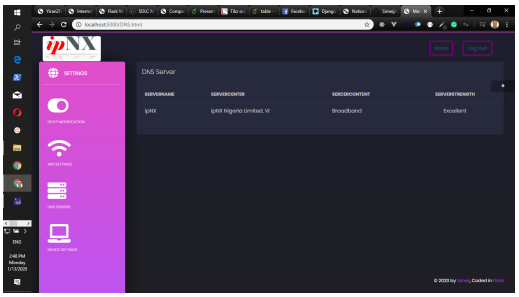
(a) vCPE WiFi management page



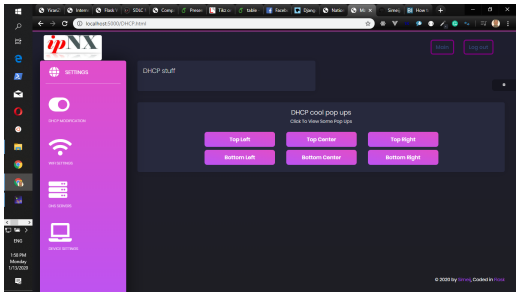
(b) vCPE Main page



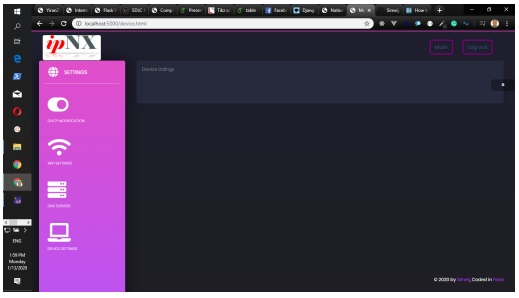
(c) vCPE DHCP page



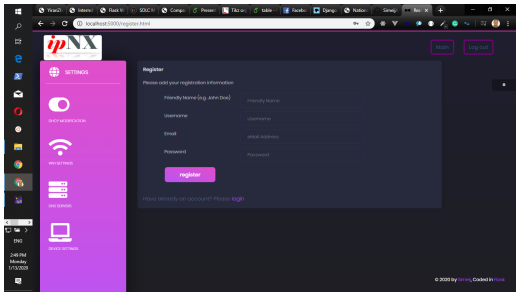
(d) vCPE DNS Server page



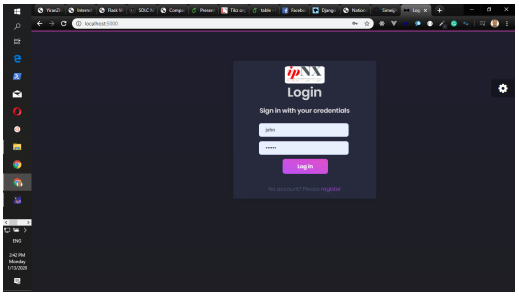
(e) vCPE DHCP page



(f) vCPE Devices List page



(g) vCPE Registration page, *restricted to an admin*.



(h) vCPE Login page

Figure 3.4: vCPE Screenshots

Dear Seun,

Please see below as requested.

S/N	Menu	Metrics	Data Visualization type	Data Source	Update Frequency	Data Availability	Data Update Frequency
1	Summary (Home Page)	Total Number of calls till date	Number Chart	3CX Call Centre solution	Real Time	Real Time	Daily
		Total number of retail customers	Number Chart	Freeside	Daily	Daily	Daily
		Active Retail customers	Number Chart	Freeside	Daily	Daily	Daily
		Inactive Retail customers	Number Chart	Freeside	Daily	Daily	Daily
		No of customers on each plans	Column Chart	Freeside	Daily	Daily	Daily
2	Retail Support Center	Total Calls	Number Chart	3CX Call Centre solution	Real Time	Real Time	Daily
		Abandoned Calls by Queue and Period (Hourly, daily, weekly, monthly or quarterly)	Column Chart	3CX Call Centre solution	Real Time	Real Time	Daily
		Average call Handle Times by Queue (Hourly, daily, weekly, monthly or quarterly)	Column Chart	3CX Call Centre solution	Real Time	Real Time	Daily
		Average call Wait Times by Queue and Period (Hourly, daily, weekly, monthly or quarterly)	Column Chart	3CX Call Centre solution	Real Time	Real Time	Daily
			Column Chart	4CX Call Centre solution	Real Time	Real Time	Daily
		Repeat Call Rate					
3	Ticketing and Resolution	Average Ticket Handle Time By Queue and Period (Weekly, Monthly, Quarterly)	Column Chart	OTRS	Daily/On Request	Daily	Daily
		SLA Violation Rate by Queue and Period (Weekly, Monthly, Quarterly)	Column Chart	OTRS	Daily/Request	Daily	Daily
		First Call Resolution Rate (Weekly, Monthly,Quarterly)	Pie Chart	OTRS	Daily/Request	Daily	Daily
		Top 5 Complaint Drivers	Column Chart	Column Chart	Daily/Request	Daily	Daily
4	Email & Escalation	Average Email Handle Time by Mailbox and Period (Monthly, Quarterly)		Mailbox (cac@ipnxnigeria.net, support@ipnxnigeria.net)	Daily	Daily	Daily
		Escalation Count by Month/Quarter(emails to Escalation@ipnxnigeria.net)	Number Chart	Mailbox (escalation@ipnxnigeria.net)	Daily	Daily	Daily
		Escalation Count by Month(emails to Escalation@ipnxnigeria.net)	Bar Chart	Mailbox (escalation@ipnxnigeria.net)	Daily	Daily	Daily
5	i-SHOP	Foot Traffic Report By Shop	Table	Manual (BackOffice, OTRS, POS Print out,)	Daily	Daily	Daily
		SOF Processing Lead Time by Period	Column Chart	OTRS (Activation), Backoffice, Sales Report	Daily	Daily	Daily
		Average Sales by Shop and Period	Bar Chart	BackOffice, Freeside	Daily	Daily	Daily
		New Sign Up Report by Shop and Period	Bar Chart	BackOffice, Sales Report	Daily	Daily	Daily
6	Social Media	Average response time	Line graph	Manual reporting (Twitter, Facebook, Instagram, LinkedIn)	Weekly	Weekly	Weekly
		Followership count trend	Line graph	Manual reporting (Twitter, Facebook, Instagram, LinkedIn)	Weekly	Weekly	Weekly
7	CX & Retention	VOTC Report	Table	Manual	Weekly	Weekly	Weekly
		Survey Report NPS	Line graph	Manual (Document)	Monthly	monthly	Twice a year
		Reimbursement Trend by Period	Line graph	Manual	Weekly	Weekly	Weekly
		Reimbursement Trend by Location	Line graph	Manual	Weekly	Weekly	Weekly
		CX Dash board	bar chart	Manual(Excel)	Weekly	Weekly	Weekly
		Community based feed back post installation					
		Activation journey stage service stage					
		Touch points ratings	bar chart	Manual(Excel)	Weekly	Weekly	Weekly
		Cssp Sign Up	Bar Chart	Freeside	Daily	Daily	Daily
		Ipnx Activation Dashboard	Bar Chart	Ftth monitor	Weekly	Weekly	Weekly

Thanks a million.

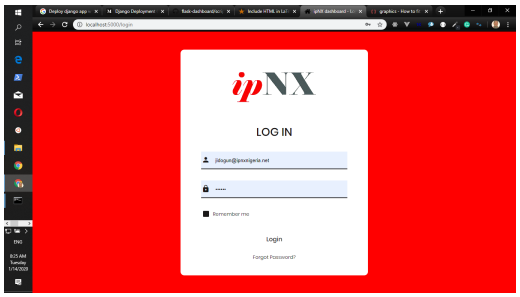
Kind Regards,

Okenla Christiana
Head, Customer Experience & Advocacy,
ipNX Nigeria Limited
10, Adenubi Close, Ikeja
Lagos, Nigeria
D/L: +234 (01) 6340002
Mobile: +234 (0) 7017770038 , 08022228644
Email: cokenla@ipnxnigeria.net
www.ipnxnigeria.net
Twitter: @ipNXTweet |Facebook: ipNX Newsroom |LinkedIn: ipNX |Instagram: @ipnxnigeria
Lagos | Abuja | Port Harcourt | Kano | Ibadan

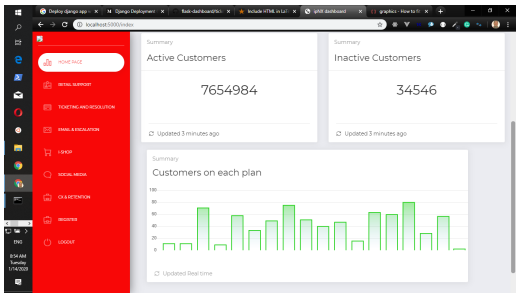
ipNX ...Leveraging technology to create innovative solutions that help mankind thrive.
On 10/2/2019 2:33 PM, Oluboyo, Charles Oluwaseun wrote:

> Hello Christiana.
>
> Please share the dashboard specification we discussed 2 weeks ago with John, keeping me in copy.
>
> Regards,
>
> 'Seun
>
> Sent from my BlackBerry 10 smartphone.
> Original Message
> From: jjdogun@ipnxnigeria.net
> Sent: Wednesday, October 2, 2019 2:16 PM
> To: soluboyo@ipnxnigeria.net
> Subject: Enquiry
>
> Good afternoon sir,
>
> How has it been? Hope everything is good.
>
> I just want to make enquiries about the dashboard project you talked about sir.

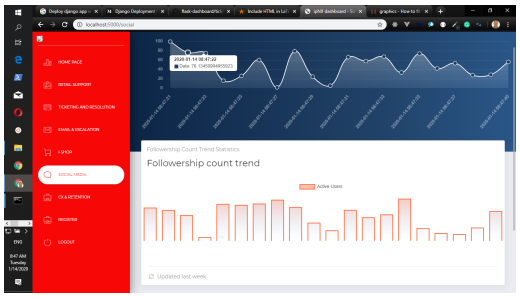
Figure 3.5: Customer Experience Analysis (CEA) Dashboard specification sent by *ip*NX's Head of Customer Experience & Advocacy.



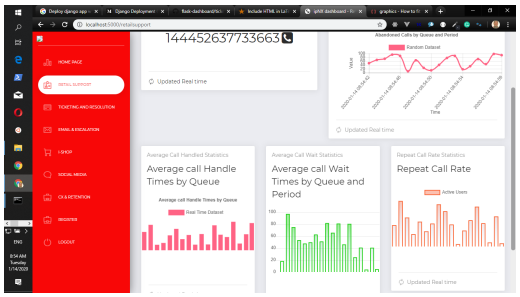
(a) CEA dashboard Login page



(b) CEA dashboard Landing or main page

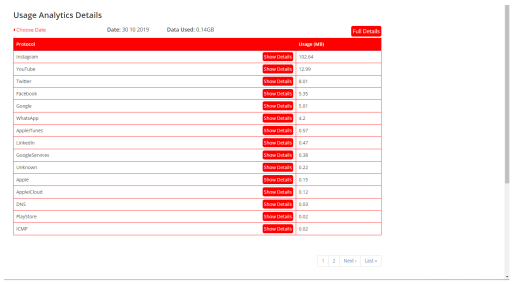


(c) CEA dashboard Social menu page

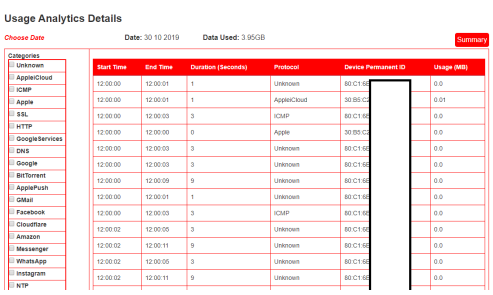


(d) CEA dashboard Retail menu page

Figure 3.6: Some CEA dashboard pages with Real-time data visualization.

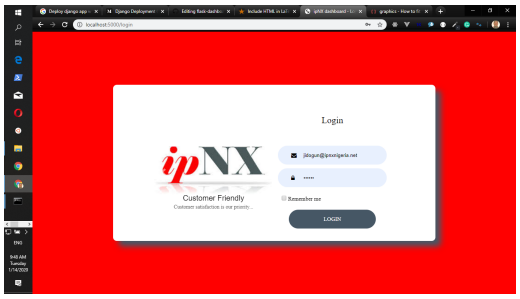


(a) DUA dashboard data analytics

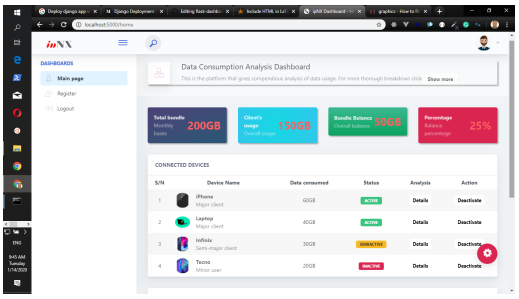


(b) DUA dashboard data analytics details

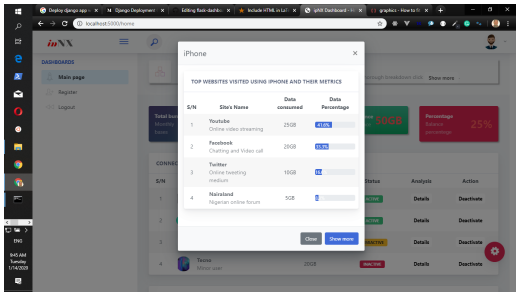
Figure 3.7: Former DUA dashboard pages.



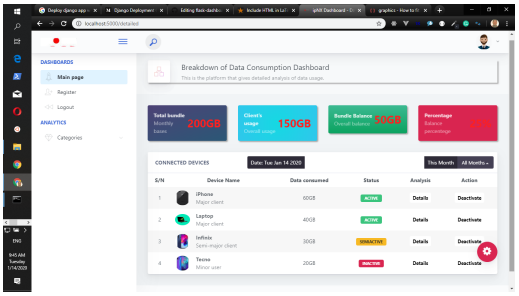
(a) DUA dashboard Animated Login page



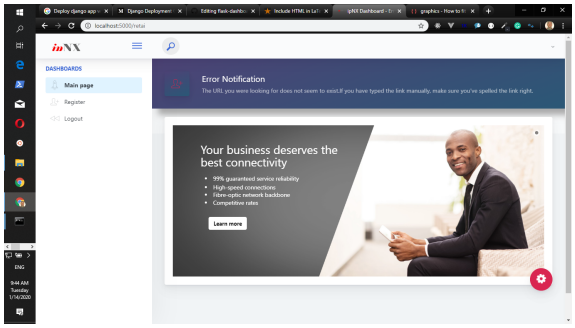
(b) DUA dashboard data analysis page



(c) DUA dashboard modal data analysis page



(d) DUA dashboard detailed data analysis page



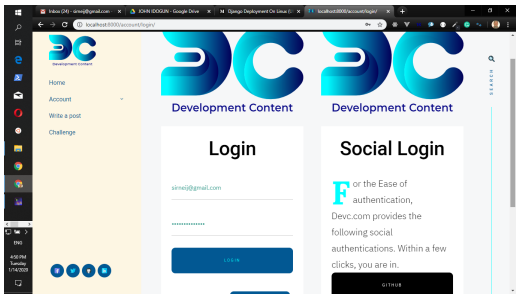
(e) DUA dashboard Animated Error page

Figure 3.8: Re-designed DUA dashboard pages.

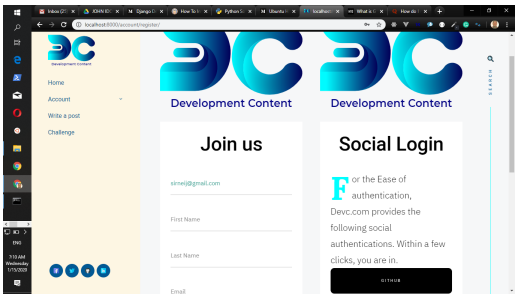
a Business Analyst and UI/UX designer in *ip*NX's department of Business Intelligence and Data Analytics (BIDA). It was fully implemented in Django framework with PostgreSQL database at the back-end and a template initially designed by Styleshout.

Devc was built with three (3) other main applications embedded, each embedding other systems in turn:

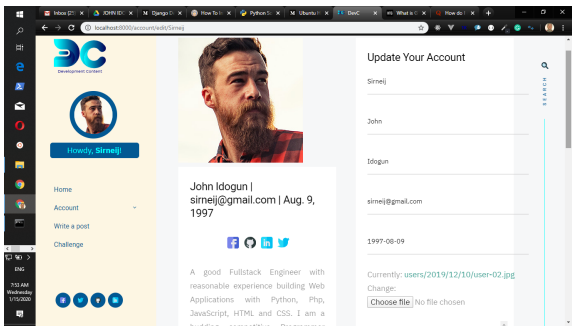
Account management system: This application provides interfaces for users registration and authentication. It allows direct authentication from popular social and technical websites such as Facebook, Twitter and Github among others as shown in Figure 3.9.



(a) **devc** Login page with social authentication



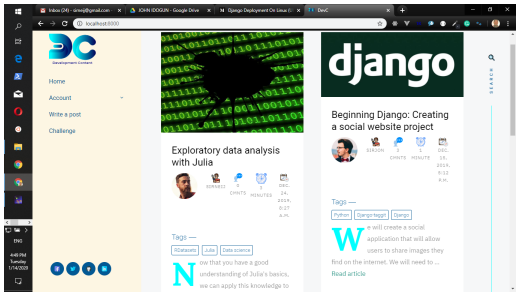
(b) **devc** Signup page with social authentication



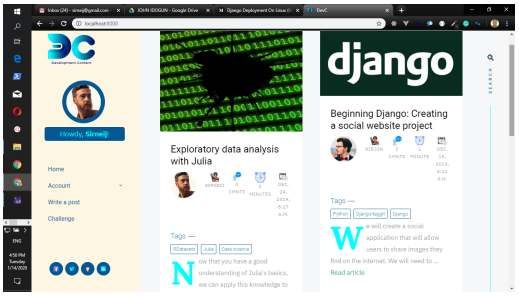
(c) **devc** Account update page

Figure 3.9: **devc** Account management system.

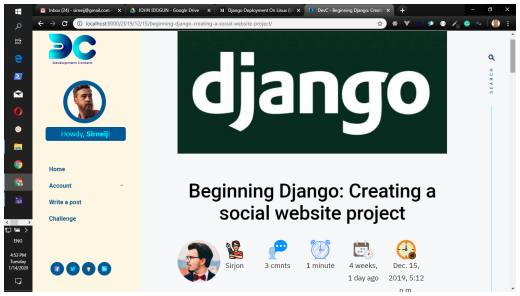
Blogging system: This is the heart of the overall system. It controls user's posts - their creation and update, with a rich text editor and advanced text formatting system; and posts' deletion - profiles and portfolio with various mini-systems such as full-text search engine, tagging, related-posts recommendation, threaded commenting and slack-like chatting systems, embedded. Users' posts could also be liked and shared asynchronously. Some of these features are shown in Figure 3.10.



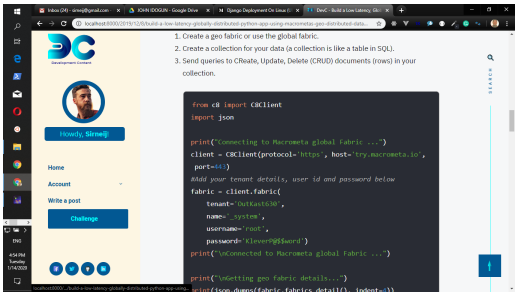
(a) **devc** main page without user authentication



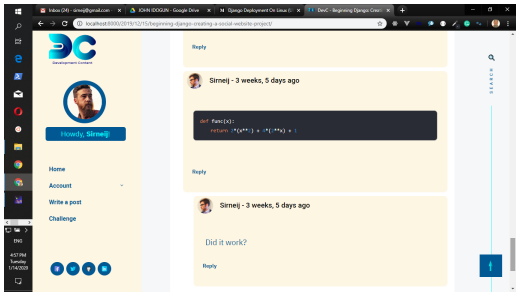
(b) **devc** main page with user authentication, recent posts shown first.



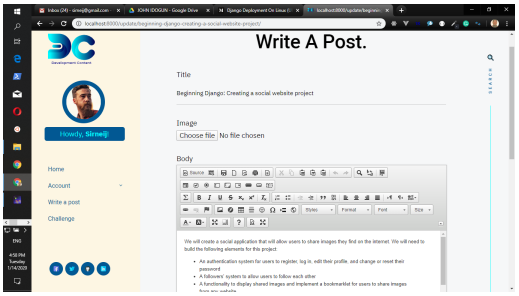
(c) **devc** Post detail page with user authentication



(d) **devc** syntax highlighting feature of the post detail page



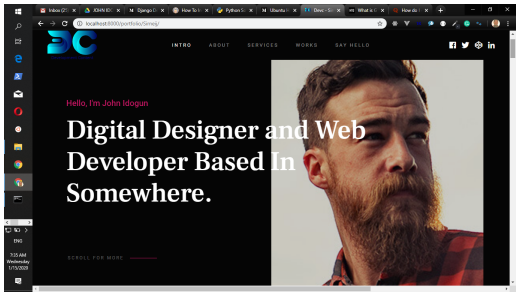
(e) **devc** Threaded commenting section of the post detail page



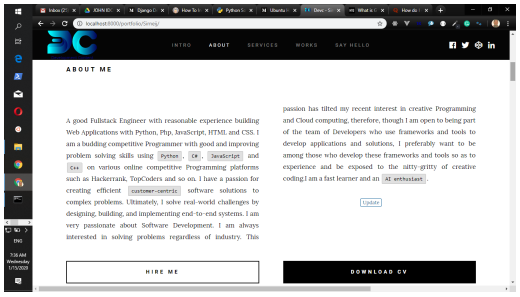
(f) **devc** post creation and update page with a Rich text editor

Figure 3.10: **devc** Blogging system.

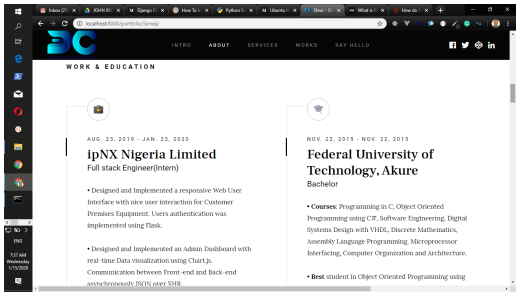
Portfolio system: This serves as an extension of the account management system where authors of post(s) showcase their skills to their potential recruiters. It structures authors profiles, education, skills, projects and recommendations accordingly which can be downloaded in Portable Document Format (PDF). Figure 3.11 gives a quick tour of some of these features.



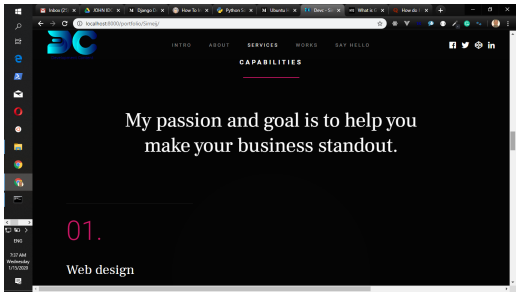
(a) **devc** portfolio introduction section.



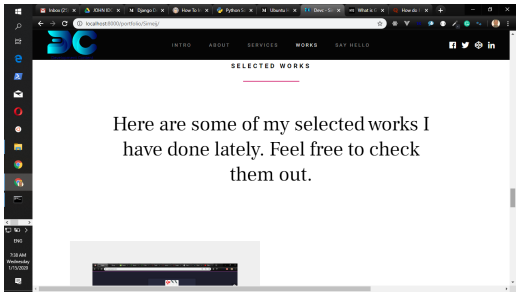
(b) **devc** portfolio about section.



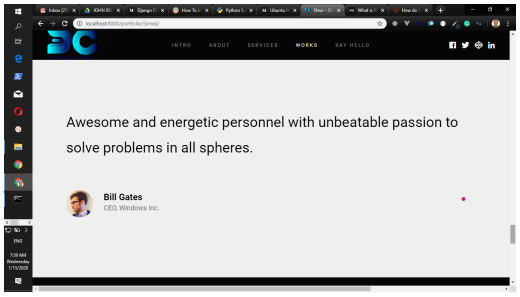
(c) **devc** portfolio experience section



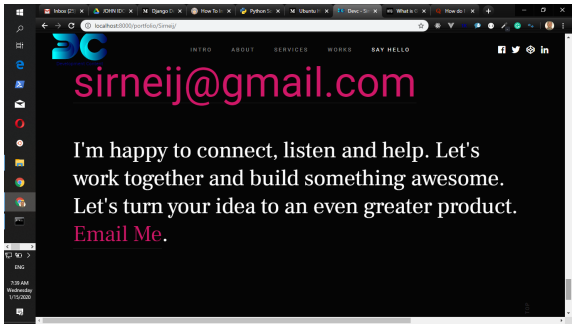
(d) **devc** portfolio service section



(e) **devc** portfolio project showcase section



(f) **devc** portfolio recommendation section



(g) **devc** portfolio Get-in-touch section

Figure 3.11: **devc** Portfolio system.

3.3 Julia-based projects

3.3.1 Brief Introduction to Julia Programming Language

Julia, a new programming language offering a unique combination of performance and productivity that promises to change scientific computing and programming in general, is an open-source language for high-performance technical computing and data science created by Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman, some of the best minds in mathematical and statistical computing, and first released in 2012.

It picks the best parts of existing programming languages, providing out-of-the-box features such as a powerful REPL, an expressive syntax, Lisp-style metaprogramming capabilities, powerful numeric and scientific programming libraries, a built-in package manager, efficient Unicode support, and easily called C and Python functions, Salceanu (2018).

3.3.2 Web Crawler

Web crawler is a program that implements web scraping, a technique for extracting data from web pages using software, and it is a pivotal component of data harvesting.

The essence of this web scraping script was to harvest some from the **devc** blog for proper usage in a Wiki game, a project yet to be concluded. Code Snippet gives the source code of the web crawler.

Code Snippet 3.9: webcrawler.jl

```
using HTTP, Gumbo
const PAGE_URL = "http://localhost:8000/"
const LINKS = String[]

function fetchPage(url)
        response = HTTP.get(url)
        # if response.status == 200 && parse(Int,
        Dict(response.headers)["Content-Length"]) > 0
        #     String(response.body)
        # else
        #     ""
        # end # if
        response.status == 200 && parse(Int,
        Dict(response.headers)["Content-Length"]) > 0 ?
        String(response.body) : ""
end # function

function extractlinks(elem)
```

```
        if isa(elem, HTMLElement) && tag(elem) == :a && in("href",
        collect(keys(attrs(elem))))
                url = getattr(elem, "href")
                #startswith(url, "/portfolio/") && push!(LINKS, url)
                startswith(url, "/2019/") && ! occursin(":", url)
                        && push!(LINKS, url)
        end
        for child in children(elem)
                extractlinks(child)
        end
end # function


content = fetchPage(PAGE_URL)
if ! isempty(content)
        dom = Gumbo.parsehtml(content)
        extractlinks(dom.root)
end # if
display(unique(LINKS))

```

3.4 Software Deployment

Software Deployment involves packaging up software products and putting them in a production environment where they can be run since having them only on the local machine barricades access to them. A production environment is the canonical version of the locally housed application and its associated data.

3.4.1 Python web application deployment

Python web application deployments are comprised of many pieces that need to be individually configured. Figure 3.12 is a map that visually depicts how each deployment topic relates to each other.

3.4.2 Deployement hosting option

According to Makai (2020), four options are available for deploying and hosting web applications, namely:

1. ”Bare metal” servers
2. Virtualized servers
3. Infrastructure-as-a-service
4. Platform-as-a-service

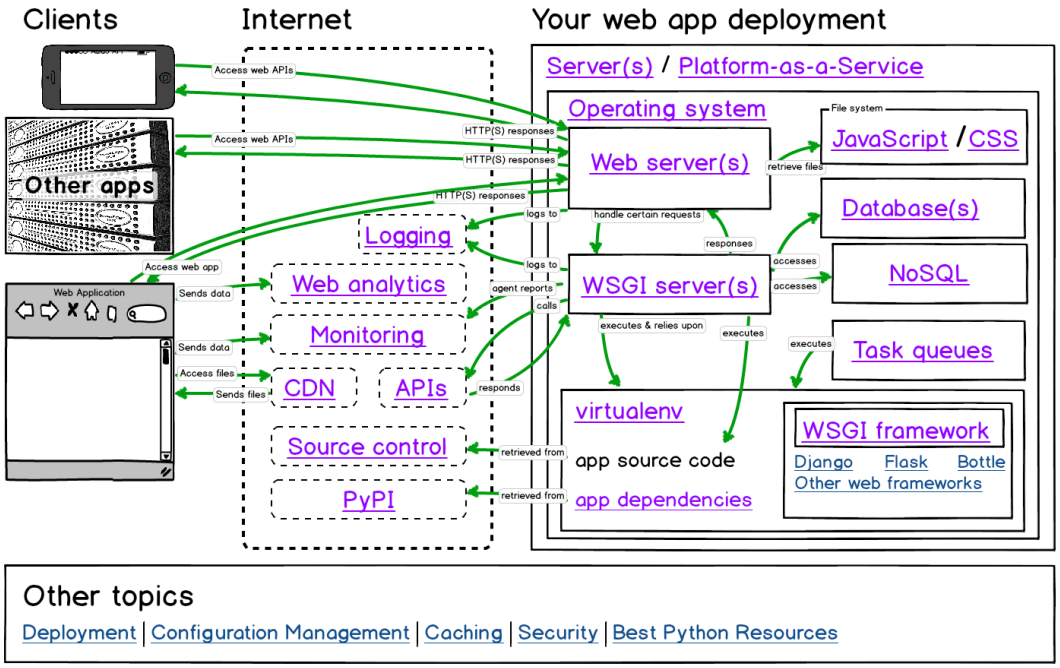


Figure 3.12: Python web application deployment map, from Makai (2020)

The first three options require the deployer to be provided with one or more servers having a Linux distribution wherein system packages, a web server, WSGI server, database and the Python environment are then installed. There then the application can be pulled from source and installed in the environment.

This was the option through which the projects built were deployed and below is a comprehensive work-through for the applications deployment.

3.4.3 Python applications deployment: General setup

- Having been provided with a remote private server based on Ubuntu 16.04 LTS with the ip address 10.50.0.20 and a password, the server was logged into using Secure Shell (SSH).

  ```
  ~$ ssh jidogun@10.50.0.20
  ```

  This required a password and was inputted appropriately.

- As it is one of the best practises to update a server when one logs in, the Linux distribution was updated.

  ```
  ~$ sudo apt-get update
  ```

- Python requires its Package manager, `pip`, to install its dependencies easily, therefore, it was installed alongside the proxy server used, **`nginx`** and **`git`**.

  ```
  ~$ sudo apt-get install python-pip nginx git
  ```

- As expected, **`nginx`** comes with a default configuration file which can be found in a **`/etc/nginx/sites-enabled/`**. This was removed to allow personalised configuration.

  ```
  ~$ sudo rm /etc/nginx/sites-enabled/default
  ```

- Having gotten rid of that default file, a personalised configuration file was created.

  ```
  ~$ sudo touch /etc/nginx/sites-available/file_name
  ```

  where `file_name` is the desired name given to the file. It was called `app_settings` in my case, so:

  ```
  ~$ sudo touch /etc/nginx/sites-available/app_settings
  ```

- Since the just created file needed to be active for its contents to be used by **`nginx`**, a symbolic file, pointing to the **`app_settings`** file created above, was made in **`/etc/nginx/sites-enabled/app_settings`**.

  ```
  ~$ sudo ln -s /etc/nginx/sites-available/app_settings
  ↪  /etc/nginx/sites-enabled/app_settings
  ```

- The symbolic file made above was edited using **`nano`** editor, to cater for all the applications to be deployed using different port numbers:

  ```
  ~$ sudo nano /etc/nginx/sites-enabled/app_settings
  ```

Code Snippet 3.10: /etc/nginx/sites-enabled/app_settings

```
server {
        listen 8004;
        location / {
                include proxy_params;
                proxy_pass http://127.0.0.1:8000;
        }
}
server {
        listen 8001;
        location / {
                include proxy_params;
                proxy_pass http://127.0.0.1:8008;
        }
}
```

```
server {
        listen 8002;
        location / {
                include proxy_params;
                proxy_pass http://127.0.0.1:8888;
                proxy_http_version 1.1;
                proxy_set_header Connection "";
                proxy_buffering off;
                chunked_transfer_encoding off;
        }
}
server {
        listen 8003;
        location / {
                include proxy_params;
                proxy_pass http://127.0.0.1:8999;
        }
}
```

Four server instances were created since four applications were deployed. The first instance pointed to Django blog, since the default port for django applications is 8000. The rest instances were used for the three dashboards written in flask.

- Next, `nginx` was restarted.

```
~$ sudo systemctl restart nginx
```

With the above configurations, everything regarding `nginx` was pretty much done with. The next inline was to deal with framework-specific configurations.

3.4.4 Framework-specifics

Before getting started with the framework-specific configurations, it was deemed necessary to organize the applications based on frameworks. Thus, different folders were created.

```
~$ mkdir flask_app django_app
```

Obviously, all flask-based applications would take `flask_app` directory as their habitat while `django_app` would house django-based applications. The applications were cloned from their respective github repositories into either of these folders as appropriate.

Flask

First off, all flask-based applications were cloned into `flask_app` directory.

```
~/flask_app$ git clone https://github.com/Sirneij/CPE-dash.git
~/flask_app$ git clone https://github.com/Sirneij/CEA-dash.git
~/flask_app$ git clone https://github.com/Sirneij/DUA-dash.git
```

Thence, the undermentioned operations were performed on each of them, using `CPE-dash` as an instance for clarification.

- First off, directory was changed to `CPE-dash`

  ```
  ~/flask_app$ cd CPE-dash
  ```

- A virtual environment for the application must be created to avoid Python version conflict issues. But First, **`virtualenv`** was installed.

  ```
  ~/flask_app/CPE-dash$ sudo pip install virtualenv
  ```

- Next, a virtual environment was created for it, specifying the python version to be used(since all the applications required at least python3.6 to function optimally, it was selected as the python version to be used).

  ```
  ~/flask_app/CPE-dash$ virtualenv -p /usr/bin/python3.6 cpe #Environment name.
  ```

- Having created the virtual environment, it was activated.

  ```
  ~/flask_app/CPE-dash$ source cpe/bin/activate
  ```

- That done, the application's dependencies, stored in **`requirements.txt`** file, and the real Python server, **`gunicorn`** were installed.

  ```
  (cpe):~/flask_app/CPE-dash$ sudo pip install -r requirements.txt gunicorn
  ```

- Penultimately, the applications `run.py` was edited to reflect the host and port configured in **`/etc/nginx/sites-enabled/app_settings`**, leaving the first server instance for django.

  ```
  (cpe):~/flask_app/CPE-dash$ sudo nano run.py
  ```

  Code Snippet 3.11: (cpe): /flask_app/CPE-dashapp/run.py

```
from app import create_app
from app import db
app = create_app()
db.create_all(app=create_app())
if __name__ == '__main__':
        app.run(host="0.0.0.0", port=8008, debug=False)
```

- Finally, `gunicorn` was called and subsequently served with the main app's file name(without .py extension), in this case, `run`

  ```
  (cpe):~/flask_app/CPE-dash$ gunicorn run:app -b 127.0.0.1:8008 &
  ```

  That was it! Application was successfully deployed! This procedure was repeated to deploy the other two flask apps.

To confirm, a browser was initiated and the ip address of the server provided as well as the port the app was listening to, say **10.50.0.20:8001** (for CPE-dash), was inputted as Uniform Resource Locator (URL) or web address and it returned a full version of the initial app.

Django

The steps observed in deploying flask-based and Django-based applications are almost the same aside some of the dependencies that django requires. Also, the developed blog used PostgreSQL database instead of SQLite which must be installed and configured specially for the application to execute smoothly.

Getting started, PostgreSQl and its dependencies were installed.

```
~$ sudo apt-get install python3-dev libpq-dev postgresql
↪  postgresql-contrib
```

CAVEAT: It's extremely important to successfully install `python3-dev` and `libpq-dev` as they are the build prerequisites which must be met in order to install `Psycopg`, a PostgreSQL adapter for the Python programming language, from a source distribution package. `Python3-dev(or python-dev)` provides the header files and its absence results in *error: Python.h: No such file or directory* when installing psycopg or psycopg2 as the case may be. `libpq` header files are provided in the `libpq-dev` package. *error: libpq-fe.h: No such file or directory* is thrown if the package is not installed.

If the above command did not install these dependencies properly, use:

```
~$ sudo apt-get install python3.6-dev libpq-dev postgresql
↪  postgresql-contrib #specify the python version used
```

Next, a database user and database for the Django app were created:

```
~$ sudo su - postgres #changes to postgres environment
postgres@10.50.0.20:~$ createuser --interactive -P
Enter name of role to add: devc #adds devc
Enter password for new role:
```

```
Enter it again:
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) n
Shall the new role be allowed to create more new roles? (y/n) n
```

The commands about created a database user devc and assigned roles to it.

```
postgres@10.50.0.20:~$ createdb --owner devc django_app_db #creates
↪  django_app_db using devc user
```

Having added a user and a database, logging out of the postgres environment was done.

```
postgres@10.50.0.20:~$ logout
```

Directory was changed into django_app and the app was cloned from github. Thence, directory was changed into the cloned app directory.

```
~$ cd django_app
~/django_app$ git clone https://github.com/Sirneij/devcAll.git
~/django_app$ cd devcAll
```

The settings.py file of the app located in /devc/devc directory was modified to configure the database settings created above. The host was also set.

```
~/django_app/devcAll$ nano /devc/devc/settings.py
```

Code Snippet 3.12: /devc/devc/settings.py

```
1  ...
2  DATABASES = {
3          'default': {
4                  'ENGINE': 'django.db.backends.postgresql_psycopg2',
                   ↪  'NAME': 'django_app_db',
5                  'USER': 'devc',
6                  'PASSWORD':'',#input your password
7                  'HOST': 'localhost',
8                  'PORT': '',
9          }
10 }
11
12 ALLOWED_HOSTS = ['*']#you could have input your ip address or domain name
   ↪  instead. '*' is for simplicity.
13 ...
```

To effect the changes made, migrations were made, super user was created and static files were collected.

```
~/django_app/devcAll$ cd devc #changes directory to devc where
↪  manage.py is located
~/django_app/devcAll/devc$ python manage.py migrate  #Syncs the
↪  database.
~/django_app/devcAll/devc$ python manage.py collectstatic #collects
↪  the static files
(devcenv):~/django_app/devcAll$ python manage.py createsuperuser
↪  #creates the super user
Username(or 'jidogun' if not specified):
Email:
Password:
Password again:
```

Django and PostgreSQl specifics concluded. Virtual environment was created and activated as demonstrated and specified under Flask app deployment above and requirements as well as gunicorn were installed and instantiated!

```
~/django_app/devcAll$ virtualenv -p /usr/bin/python3.6 devcenv
↪  #Creates devcenv virtual environment using python3.6 as
↪  interpreter.
~/django_app/devcAll$ source devcenv/bin/activate #activates virtual
↪  environment
(devcenv):~/django_app/devcAll$ sudo pip install -r requirements.txt
↪  gunicorn #installs dependencies and gunicorn
(devcenv):~/django_app/devcAll/devc$ gunicorn devc.wsgi -b
↪  0.0.0.0:8000 & #serves gunicorn the app persistently.
```

The applications could now be accessed online...

If error of the below form erupted:

```
[2020-01-20 15:08:02 +0000] [19640] [INFO] Starting gunicorn 19.4.5
[2020-01-20 15:08:02 +0000] [19640] [ERROR] Connection in use: ('0.0.0.0', 8000)
[2020-01-20 15:08:02 +0000] [19640] [ERROR] Retrying in 1 second.
[2017-01-20 15:08:03 +0000] [19640] [ERROR] Connection in use: ('0.0.0.0', 8000)
[2020-01-20 15:08:03 +0000] [19640] [ERROR] Retrying in 1 second.
[2020-01-20 15:08:04 +0000] [19640] [ERROR] Connection in use: ('0.0.0.0', 8000)
[2020-01-20 15:08:04 +0000] [19640] [ERROR] Retrying in 1 second.
[2020-01-20 15:08:05 +0000] [19640] [ERROR] Connection in use: ('0.0.0.0', 8000)
```

Below commands would get them solved:

```
$ sudo netstat -nlp | grep :80 #finds the process using port 80
$ sudo fuser -k 8000/tcp #kills the process
```

```
#Reserve gunicorn with the application

(devcenv):~/django_app/devcAll/devc$ gunicorn devc.wsgi -b 0.0.0.0:8000 & #serves
↪  gunicorn the app persistently.
```

Chapter 4

Conclusion and Recommendation

4.1 Conclusion

During this internship, I had the chance to work on many aspects of Software Engineering ranging from the Planning and Requirement Analysis of software products to their actual development and eventual Deployment. I developed new technical knowledge and soft skills while improving or enhancing previous ones in the course of implementing the robust projects undertaken.

- I gained new knowledge in the area of Working with PostgreSQL - for full-text search capabilities - and SQLite Databases, regarding the various issues involved and mechanisms in these systems.
- I brushed up my knowledge of Python and JavaScript, as they were required to implement the projects.
- It was a challenging but awesome experience deploying applications to a Linux server from the ground up. Countless times of breaking and fixing server dependencies were worth the stress after all as the process exposed nifty things about systems administration.
- The importance and huge impact portability and scalability have on decisions about hardware, technologies/tools to be used, system architecture, algorithms, and so on were duly learnt and observed.
- Extensive exposure to Git, as a version control system, for safe storage of codes and projects.
- Significant knowledge of using asynchronous programming for some background tasks to provide better user experience was well learnt.

- Hands-on experience engaging in cross-platform troubleshooting of software products.
- Total exposure to the Linux operating system, especially Ubuntu 14.04 LTS and 16.04 LTS, for software deployment using Apache2, nginx and gunicorn.

Some theoretical knowledge gained in school proved helpful and useful such as CPE407(Software Development Techniques), a useful ingredient in following along with the software development processes at *ip*NX Nigeria Limited, and CPE202(Programming in C Language) as well as CPE302(Object Oriented Programming using C#), for laying a good foundation in programming to pick up other languages effortlessly.

4.2 Recommendation

There is no gainsaying the fact that the idea of Industrial Training is laudable and its role in equipping students with relevant and soft skills cannot be overemphasized. Out of the various things learned and experiences gained, the undermentioned are worth taking into consideration:

1. **Adoption and Enforcement of Project Based Learning (PBL)**: Project Based Learning (PBL) is a pedagogy in which students learn by actively engaging in real-world and personally meaningful projects. In this method, students work on a project over an extended period of time – from a week up to a semester – that engages them in solving a real-world problem or answering a complex question.

   As a result, students develop deep content knowledge as well as critical thinking, collaboration, creativity, and communication skills. Project Based Learning unleashes a contagious, creative energy among students and teachers. Using and enforcing this method, with proper and decent design, will help narrow the wide gap between University Education and Industry required knowledge.

2. **Incorporation of a full course on Algorithms and Data structures**: Data structure and associated algorithms are an important part of any computer science or engineering curriculum as they expose students to critical and advanced thinking skills which will ultimately ameliorate problem-solving prowess. It is highly important that students are aware of them because most Interview questions are based on their application.

3. **Use of innovative approaches to teaching and learning**: The use of innovative approaches to teaching and learning is very essential such as the use of teaching

aids, ensuring the availability of teaching materials online before and after lectures among others.

4. The Federal Government should make it an obligation for all firms to absorb students on Industrial training and ensure that such students are well-trained. This stretches out of how daunting and herculean it is to secure a placement by most students.

References

Bayer, M. (2016). *SQLAlchemy Documentation*. Release 0.9.10.

Grinberg, M. (2014). *Flask Web Development*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, second edition.

Holovaty, A. and Willison, S. (2019). *Django Documentation*. Django Software Foundation. Release 3.1.dev.

Makai, M. (2020 (accessed January 15, 2020)). *Deployment*. Full Stack Python. https://www.fullstackpython.com/deployment.html.

Melé, A. (2018). *Django 2 by Example: Build powerful and reliable Python web applications from scratch*. Packt Publishing Ltd., 35, Livery Street, Birmingham B3 2PB, United Kingdom.

Pal, S. K. (2020 (accessed January 12, 2020)). *Software Engineering | Phases of Prototyping Model | Set - 2*. https://www.geeksforgeeks.org/software-engineering-phases-prototyping-model-set-2/.

Ronacher, A. (2010 (accessed January 9, 2020)). *Flask*. https://palletsprojects.com/p/flask/.

Ronacher, A. (2017). *Jinja2 Documentation*. Release 2.9.6.

Salceanu, A. (2018). *Julia Programming Projects*. Packt Publishing Ltd, 35, Livery Street, Birmingham B3 2PB, United Kingdom, first edition.

Sharma, P. and Singh, D. (2015). Comparative study of various sdlc models on different parameters. *International Journal of Engineering Research*, 4(4):188–191.

Team, E. (2017 (accessed January 12, 2020)). *SDLC Models Explained: Agile, Waterfall, V-Shaped, Iterative, Spiral*. https://existek.com/blog/sdlc-models/.

Yang, J. (2020 (accessed January 13, 2020)). *Ajax*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX.

Zhou, Y. (2010). Project report of building course management system by django framework. Report P-5, Simon Fraser University.