



# A CLOUD-BASED C/C++ COMPILER FOR SMART DEVICES

A Thesis Presented to the Department of  
Computer Science

African University of Science and Technology

In Partial Fulfilment of the Requirements for the Degree of  
Master of Science in Computer Science

By

Egwim Christian Nnaemeka

NOVEMBER, 2017.

**CERTIFICATION**

This is to certify that the thesis titled “A CLOUD-BASED C/C++ COMPILER FOR SMART DEVICES” submitted to the school of postgraduate studies, African University of Science and Technology (AUST), Abuja, Nigeria for the award of the Master's degree is a record of original research carried out by Egwim Christian Nnaemeka in the Department of Computer Science.

A CLOUD-BASED C/C++ COMPILER FOR SMART DEVICES

By

Egwim Christian Nnaemeka

A THESIS APPROVED BY THE COMPUTER SCIENCE DEPARTMENT

RECOMMENDED:

---

Supervisor, Professor Mohamed Hamada

---

Head, Department of Computer Science

APPROVED:

---

Chief Academic Officer

---

Date

© 2017

Egwim Christian Nnaemeka

ALL RIGHTS RESERVED

## **ABSTRACT**

The foundation of many of today's programming languages is the C/C++ language, due to its success and directives that span across several programming paradigms, conceptual models and runtime environments. Smart devices are getting more powerful, cheaper and hence more popular as seen in ubiquitous computing and internet of things (IOT). The goal of this project is to design and develop an integrated development environment (IDE) for C/C++ Programming language to run on smart devices, so that the user can edit,

develop, debug, compile, and run C/C++ programs on smart devices. The project also includes an implementation of a cloud-based server that hosts the compiler.

## ACKNOWLEDGEMENTS

I take this opportunity to express my profound gratitude and deep regards to the Almighty GOD for HIS exemplary guidance, monitoring and constant encouragement throughout the course of my M.Sc. program.

I also take this opportunity to express a deep sense of gratitude to **Professor Mohamed Hamada** for his cordial support, valuable information and guidance, which helped me in completing this task through various stages.

I am obliged to staff members of this great department, for the valuable information provided by them in their respective fields. I am grateful for all their cooperation during the period this program.

Lastly, I thank my parents, beautiful fiancée, brothers, sisters and friends for their constant encouragement and visitation without which this project would not be possible.

## DEDICATION

This thesis is dedicated to my beloved parents Mr. Emmanuel Egwim and Mrs. Lucy Egwim and the queen of my heart my fiancée Glory Benjamin.

# TABLE OF CONTENTS

ABSTRACT .....	iv
ACKNOWLEDGEMENTS.....	vi
DEDICATION.....	vi
LIST OF FIGURES .....	viii
CHAPTER ONE INTRODUCTION .....	1
1.1 Background to study .....	1
1.2 Problem statement .....	3
1.3 Aim and objectives .....	3
1.4 Expected results .....	3
CHAPTER TWO LITERATURE REVIEW .....	3
2.1 Compilers.....	4
2.1.1 Compiler architecture.....	4
2.1.2 Phases of a compiler.....	5
2.2 Concept of cloud computing.....	6
2.2.1 Types of cloud computing.....	8
2.3 Location of the cloud.....	8
2.3.1 Classification based upon service provided .....	9
2.4 Android operating system.....	10
2.5 Review of existing works .....	12
2.5.1 A cloud-based Java compiler for smart devices .....	12
2.5.2 Project implementation .....	12
2.5.3 Limitation of the system.....	12
2.5.4 Online C/C++ compiler using cloud computing .....	13
2.5.5 Cloud Compiler Based on Android .....	15
2.5.6 Cloud-based “C - Programming” Android application framework.....	18
2.6 Proposed solution to limitations of the existing works .....	21
CHAPTER THREE RESEARCH METHODOLOGY.....	21
3.1 System analysis.....	21
3.2 Analysis of the proposed system .....	22
3.3 Ionic framework.....	22
3.4 CPP.SH online compiler .....	23

3.5 System requirements analysis.....	23
3.6 Data flow diagram (DFD).....	24
3.7 Context level data flow diagram.....	25
3.8 System design .....	26
3.8.1 Entity relation model.....	26
3.8.2 Entity relationship diagram (ERD) .....	26
3.8.3 Relational database model.....	27
3.8.4 Relational database model of the proposed system.....	28
3.9 Database file design.....	28
3.10 Modules of the proposed system .....	31
CHAPTER FOUR IMPLEMENTATION OF THE SYSTEM .....	31
4.1 Tools used.....	31
4.2 Command line interface.....	32
4.3 GIT.....	33
4.4 Home page .....	34
4.5 The menu bar page.....	36
4.6 The audio lectures page .....	37
4.7 The video lectures page .....	38
4.8 The online compiler page .....	38
4.9 The project information view .....	40
CHAPTER FIVE SUMMARY, CONCLUSION AND RECOMMENDATIONS .....	40
5.1 Summary.....	41
5.2 Conclusion .....	41
5.3 Limitations of the system.....	41
5.4 Recommendations and future work .....	42
REFERENCES .....	42
APPENDICES .....	45

## LIST OF FIGURES

Figure 1.1: Anatomy of a compiler .....	2
-----------------------------------------	---



Figure 2.1: Compiler architecture .....	4
Figure 2.2: Compiler components .....	5
Figure 2.3: Phases of a compiler .....	6
Figure 2.4: Cloud Service .....	10
Figure 2.5: Layers of the Android OS.....	12
Figure 2.6: Architecture of the project .....	15
Figure 2.7: Architecture of the system .....	18
Figure 2.8: Dataflow diagram of the system .....	18
Figure 2.9: System Architecture .....	20
Figure 2.10: Data flow diagram of the system .....	21
Figure 3.1: Context level data flow diagram .....	26
Figure 3.2: Entity-relationship diagram .....	28
Figure 3.3: Android user table .....	30
Figure 3.4: iOS user table .....	30
Figure 3.5: Mobile app table .....	30
Figure 3.6: Admin Table .....	31
Figure 3.7: Relational database design .....	31
Figure 4.1: The command line interface .....	34
Figure 4.2: GIT .....	35
Figure 4.3: Home page .....	36
Figure 4.4: Menu bar page .....	37
Figure 4.5: Lecture slide page .....	38
Figure 4.6: The audio lecture page.....	39
Figure 4.7: Video lecture page .....	40
Figure 4.8: Online compiler page .....	41
Figure 4.9: “About” page .....	42

# CHAPTER ONE INTRODUCTION

In this chapter we introduce learning technologies in the 21st century and how they affect learning among students. We also discuss mobile technology, mobile learning and multimedia learning systems. The chapter also discusses the increased use of smart mobile devices, the future of learning technologies and how we can leverage them to improve the use of mobile learning. These discussions lead us to formulate and present our problem statement as well as our aims and objectives.

## 1.1 Background to study

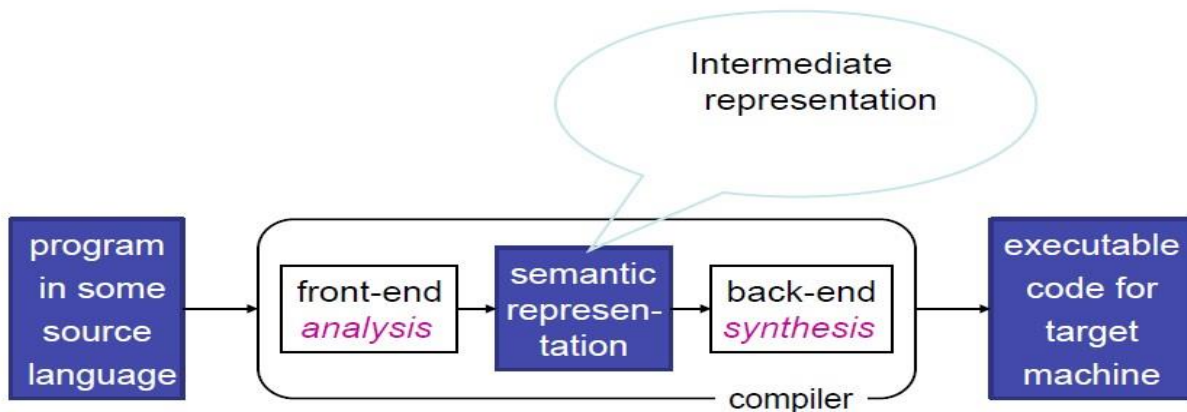
Learning technologies, especially mobile learning in schools, are changing fundamentally. Back in the day when it was necessary to ring a school bell before lectures could start, or as a researcher you needed to physically go to laboratory to run simulations or submit a job to a computer that was as large as a default TV screen size to now where it possible to receive the same quality of learning without leaving your comfort zone by means of a virtual learning platform that leverages on the concept of cloud computing.

Cloud computing allows for convenient, on-demand network access to a shared pool of configurable computing resources as either software as a service (SaaS), as platform as a service (PaaS), or as infrastructure as a service (IaaS) thereby increasing the demand for technological services in our schools to be on the increase as it provides a deployment model for these demands on a pay-as-you-use basis.

This poses a necessity not only to keep pace with the ever-rising demand, but also to be able to make these resources accessible on any device that can connect to the internet, among which are slates, pads, tablets, smartphones, notebook and desktops. It is obvious today that the use of mobile or smart devices among students and faculty outweighs desktop computer usage among them. Hence the need to provide cloud services that can run independently on any internet-enabled device and most importantly on these smart devices.

Some of the big problem areas of smart devices are seen in memory management and power consumption. An app might be spinning too many threads and soaking up many resources and even though recent smart devices use high speed processors, with clock frequencies of about 1 GHz, power consumption is still an issue. Cloud-based computation now offers a solution by making computations offline on the device but online in the cloud via internet connectivity, leading to capability boost on these devices through a tremendous leap in functionality and the amount of data apps can access.

A compiler which is the heart of any computer system because of its wide applicability in data structures and algorithms, formal languages, computer architecture and better understanding of programming language concepts, reads a program written in one language and translates it into another language. The anatomy of a compiler usually consists of front-end analysis, semantic representation and a back-end synthesis.



**Figure 1.1:** Anatomy of a compiler

Many other programming languages have been influenced by C/C++ programming language. For example, Java, C#, Python, PHP, Perl etc. are C-like in nature as a result of its directives that span across several of these languages, conceptual models and runtime environment, hence the reason for the choice of the language in this research work.

## **1.2 Problem statement**

To date, 44% of the world's population owns smart devices, up from about 10% in 2011. These smart devices are also getting more powerful, cheaper and hence more popular. It is therefore expedient to integrate a C/C++ compiler in a learning system that entails a C/C++ lecture slides component, audio lecture component, video lecture component and more importantly a cloud-based C/C++ compiler that enables users to compile and run their programs on smart devices.

This research also includes an implementation of a cloud-based server that hosts the compiler online on a remote server.

## **1.3 Aim and objectives**

The main aim of this research is to design and develop an Integrated Development Environment (IDE) for C/C++ programming language. The IDE will support smart devices such as Android-based and iOS-based devices.

## **1.4 Expected results**

- An IDE for C/C++ programming language;
- An Android mobile app containing a cloud-based C/C++ compiler component that will enable users to compile and run their programs on their smart devices; and
- An implementation of a local (or cloud) based server to host the C/C++ compiler.

# **CHAPTER TWO LITERATURE REVIEW**

This chapter discusses certain concepts relating to cloud-based compilers, which include compilers and how they work, cloud computing and the Android operating system.

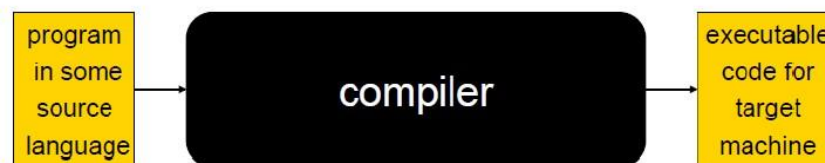
This chapter also reviews existing work relating to cloud-based compilers, their weaknesses and how the system to be developed intends to solve these weaknesses.

## 2.1 Compilers

Computer programs are formulated in a programming language and specify classes of computing processes. Computers, however, interpret sequences of particular instructions, but not program texts. Therefore, the program text must be translated into a suitable instruction sequence before it can be processed by a computer. This translation can be automated, which implies that it can be formulated as a program itself. The translation program is called a compiler, and the text to be translated is called source text, or source code [13].

### 2.1.1 Compiler architecture

Compilers are used to compile programs and convert them from a written program to executable binaries. In other words, a compiler is a program that reads a program written in one language and translates it into another language. The compiler creates executable files which can then be run in order to execute the program and its instructions, as shown in figure 2.1.



**Figure 2.1:** Compiler architecture

Every compiler primarily consists of two major components:

1. The front end: This checks the semantics and syntax of the higher-level code (written by the user). Other functions like type checking and error reporting are also performed by the front end.
2. The back end: This is the part where the translation of the language actually takes place and performs the optimal optimization through removal of redundant code or relocation of computation depending on the context. See figure 2.2 below.

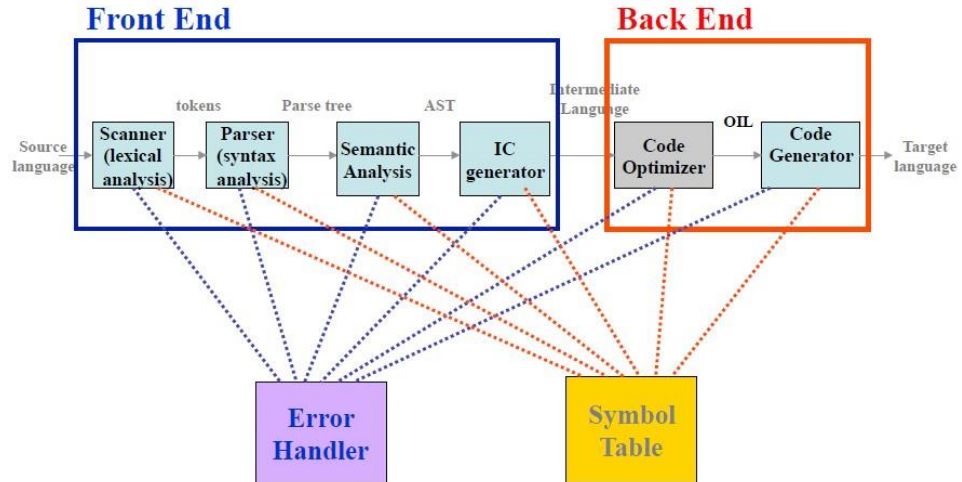


Figure 2.2: Compiler components

### 2.1.2 Phases of a compiler

The compiler has a number of phases plus a symbol table manager and error handler. This modularization is typical of many real compilers. The authors in [11] and [14] describe the phases of a compiler which are summarized in Figure 2.3 below.

S/N	PHASE	DESCRIPTION
1.	Lexical Analyzer	Break the source file into individual words, or <i>tokens</i>
2.	Syntax Analyzer	Analyze the phrase structure of the program
3.	Semantics Analyzer	Build a piece of <i>abstract syntax tree</i> corresponding to each phrase. Determine what each phrase means, relate uses of variables to their definitions, check types of expressions and request translation of each phrase
4.	Intermediate Code Generator	Transforms parse tree into intermediate language which represents source code program
5.	Code Optimizer	Optimizes intermediate codes and produces fast running machine codes
6.	Code Generator	Produces re-locatable machine codes or assembly codes
7.	Target Language	

Figure 2.3: Phases of a compiler

- The purpose of the symbol table is to store information about the occurrence of various entities such as variable names, functions etc.

- The purpose of the error handler is take care of exceptions in a system, function or even a variable.

## 2.2 Concept of cloud computing

According to [15] cloud computing refers to “flexible self-service, network-accessible computing resource pools that can be allocated to meet demand”. Services are flexible because the resources and processing power available to each can be adjusted “on the fly” to meet changes in need or based on configuration settings in an administrative interface, without the need for direct IT personnel involvement. These resources are assigned from a larger pool of available capacity (for example memory, storage, CPUs) as needed, allowing an organization to spin up a proof-of-concept application, expand that to a full prototype, and then roll it out for full use without having to consider whether existing hardware, data centre space, power and cooling are capable of handling the load. Cloud computing allows the allocation of resources to be adjusted as needed, creating a hardware-independent framework for future growth and development.

Almost anything can be hosted in the cloud, from databases and applications to complete virtual infrastructures encompassing data storage, networking and all components of the server environment. The cloud can also host virtualized user desktop environments available from any networked client device, whether or not the client has sufficient local resources to host the virtualized desktop environment and its various applications.

Google Gmail and Google Apps demonstrate the early power of cloud computing, starting in 2006; now cloud computing goes beyond simply hosting a website or database service on a machine located in a remote data centre, Cloud computing solutions have several common characteristics, regardless of their form [15]. They include:

**Managed by the provider** – cloud computing services are managed by the cloud provider. Once applications and services have been moved to external cloud computing, an organization no longer needs

to worry about local data centre issues regarding power, space and cooling, and developers need only know whether their applications will be running on one cloud service platform or another [15].

**Flexible resource assignment** – the capacity and resources available to cloud computing services can be increased or decreased, with costs adjusted according to actual consumption. This allows an organization to spin up a new offering with only minimal costs for the resources used and then to meet spikes or cyclic use patterns with increased capacity, paying for only the level of use needed. Traditional data centres must always plan for future growth, and a sudden success for a web-based offering can rapidly overrun available server and network capacity unless data centre managers purchase sufficient “spare” resources beforehand. Cloud computing draws resources from a pool as they are needed, based on the level of service consumption. This is similar to the way power companies supply power to individual organizations, billing each according to its individual use. For example, a new cloud application might experience a sudden increase in use following mention on a popular blog and require additional network bandwidth, data storage, server memory or CPU power to keep up with the sudden increase in demand. Traditional data centres would be limited by hardware constraints, while cloud computing alternatives can simply add CPUs or expand available database file storage up to predefined limits when needed and then shrink back after the demand for access has passed to manage on-demand costs [15].

**Network accessible** – cloud services are available via networked devices and technologies, facilitating rapid access by mobile customers and remote office locations. This provides an “anywhere, anytime” service model not possible in traditional data centres, where service downtime and local area outages in power and networking can impact uptime. Because cloud computing vendors can be located anywhere in the world, they can host organizational services from areas outside geopolitical turmoil or environmental threats. Before a hurricane, for example, a cloud service provider could transfer operations from Florida to Washington transparently to the service consumer [15].



**Sustainable** – because cloud providers can provide resources at need, it is possible to reduce power and cooling requirements during off-peak times, gaining economies of scale well beyond those available to single-tenanted hardware-based data services, which must stay on, waiting for later use. The flexibility in cloud hosting location allows providers to shift operations without disruption to consumers. They can move data centre activity seasonally to save on cooling costs or transfer operations to areas with excess power production capability, such as Iceland [15].

**Managed through self-service on demand** – after limits for resource availability are configured within the cloud provider's systems, available resource capacity can be automatically expanded or managed by the client with minimal effort. Bringing up a test server no longer requires access to the physical system, loading software, and configuring networking by hand; instead, the customer needs only to access their cloud provider and request a new resource allocation using the self-service user interface. As long as the organization's contractual limits on resources allow the addition, it is managed automatically without the need for further technical assistance [15].

### **2.2.1 Types of cloud computing**

Kirk Hausman et al. in [15] give a classification of cloud computing by location while a classification by type of services offered is given in [16]. The classifications are discussed below.

## **2.3 Location of the cloud**

Cloud computing is typically classified in the following ways:

1. **Public cloud:** The computing infrastructure is hosted by the cloud vendor at the vendor's premises. The customer has no knowledge of or control over where the computing infrastructure is hosted. The computing infrastructure is shared between many organizations.

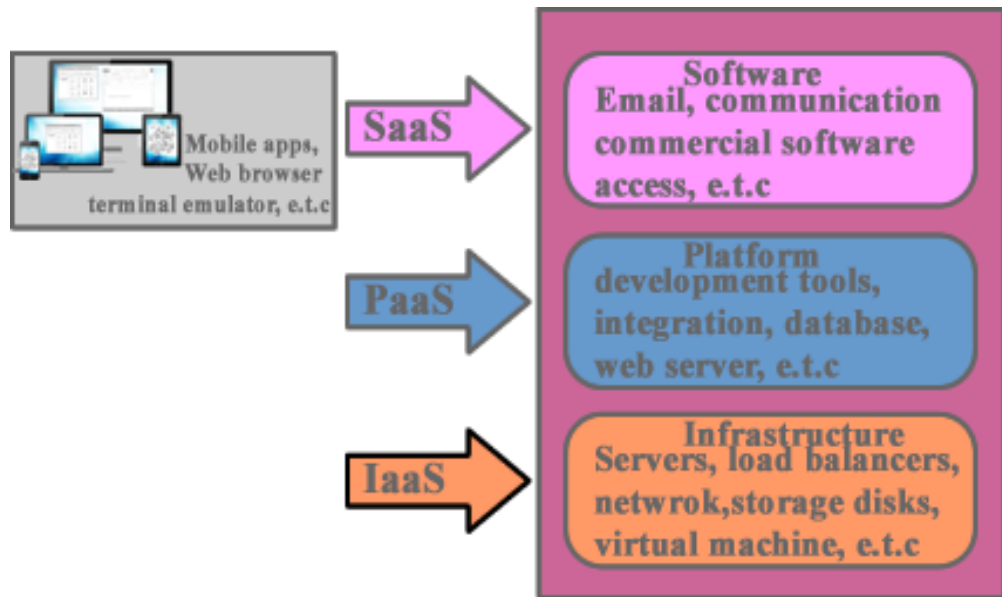
2. **Private cloud:** The computing infrastructure is dedicated to a particular organization and not shared with other organizations. Some experts consider that private clouds are not real examples of cloud computing. Private clouds are more expensive and more secure when compared to public clouds.
3. **Hybrid cloud:** Organizations may host critical applications on private clouds and applications with relatively fewer security concerns than on the public cloud. The usage of both private and public clouds together is called a hybrid cloud. A related term is cloud bursting. In cloud bursting, organizations use their own computing infrastructure for normal usage, but access the cloud using services like Salesforce cloud computing for high/peak load requirements. This ensures that a sudden increase in computing requirement is handled seamlessly.
4. **Community cloud:** This involves sharing of computing infrastructure in between organizations of the same community. For example, all government organizations within the state of California may share computing infrastructure on the cloud to manage data related to citizens residing in California.

### 2.3.1 Classification based upon service provided

Based upon the services offered, clouds are classified in the following ways:

1. **Infrastructure as a service (IaaS)** involves offering hardware-related services using the principles of cloud computing. These could include some kind of storage services (database or disk storage) or virtual servers. Leading vendors that provide IaaS are Amazon EC2, Amazon S3, Rackspace Cloud Servers and Flexiscale.
2. **Platform as a Service (PaaS)** involves offering a development platform in the cloud. Platforms provided by different vendors are usually not compatible. Typical players in PaaS are Google's Application Engine, Microsoft's Azure, Salesforce.com's force.com.
3. **Software as a Service (SaaS)** includes a complete software offering in the cloud. Users can access a software application hosted by the cloud vendor on pay-per-use basis. This is a well-established

sector. The pioneer in this field has been Salesforce.com's offering in the online customer relationship management (CRM) space. Other examples are online email providers like Google's Gmail and Microsoft's Hotmail, Google docs and Microsoft's online version of office called BPOS (Business Productivity Online Standard Suite). See figure 2.3 below:



**Figure 2.4: Cloud Service**

## **2.4 Android operating system**

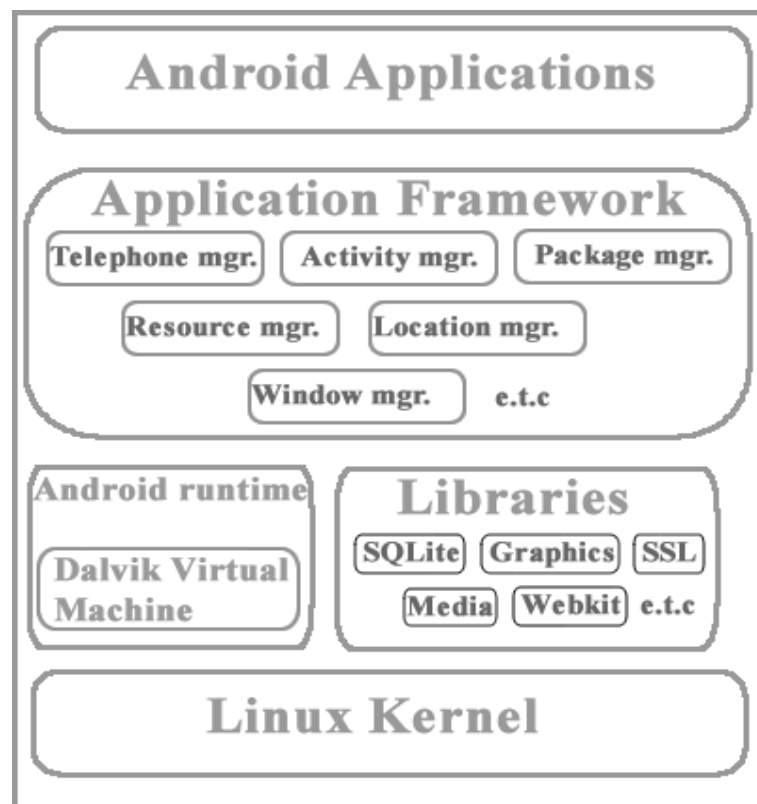
Android is a software stack for mobile devices that includes an operating system, middleware and key applications with the aim of all-time high performance by optimizing memory with faster and more accurate response. All applications are written using the Java language by enabling and simplifying the reuse of components, i.e. full access to the same framework.

In Figure 2.4 it can be seen that application programming interfaces (APIs) are used by the core applications and can be replaced or reused. They also include a set of C/C++ libraries used by components through the Android application framework. The core libraries provide most of the functionality of the Java language like data structures, utilities, file access, network access, graphics, etc.

The Dalvik Virtual Machine provides an environment in which every Android application runs in its own process, with its own instance with multiple efficient register-based VMs. The Dalvik Executable (.dex) format is optimized for minimal memory footprint and compilation.

The Linux Kernel provides threading, low memory and process management, network stack, drive model and security. Unlike PC operating systems, mobile phone operating systems are constrained by their hardware, memory, power dissipation and mobility conditions [17]. The most recent Android version is Android 6.0, known as Android Marshmallow.

Google provides the Android software development kit (SDK) for developers for developing applications for Android easily. The Android operating system provides users and developers with a complete suite of software for mobile devices such as an operating system, middleware and key mobile applications.



**Figure 2.5: Layers of the Android OS**

## **2.5 Review of existing works**

There is much research on online-based compilers. This review discusses some previous work on online compilers and how they execute. It also discusses the limitations of each.

### **2.5.1 A cloud-based Java compiler for smart devices**

In 2016 Mohammed et al. [1] developed a cloud-based Java compiler for smart devices. The aim of their project was to design and develop an IDE for Java language to run on smart devices on a chosen operating system.

### **2.5.2 Project implementation**

In developing the system, the Sphere Engine Online Compiler API was used. The system had a database component for user registration, authentication, code compilation and execution. The server-side technology used for communication with the Sphere Engine API was PHP and the database system used was MySQL.

### **2.5.3 Limitation of the system**

The major setbacks of the system developed by Mohammed in [1] are as follows:

1. Automatic focus is on the text area of the IDE on page load.
2. It can be slow when editing large files (JavaScript is not a fast language).
3. Only one syntax language is supported at the same time (no HTML and PHP syntax highlight at the same time).
4. Limited amount of compile time caused by the use of Sphere Engine.
5. No execution for GUI programs as the approach they initially opted for did not work. Though they proposed a new approach, they could not implement it because of time constraints.

#### **2.5.4 Online C/C++ compiler using cloud computing**

In 2011, Aamir, et al. [2] developed an Online C/C++ Compiler Using Cloud Computing. The aim of their project was to have a system where online programming examination with C/C++ could be conducted in their school. This project implemented the following functionalities:

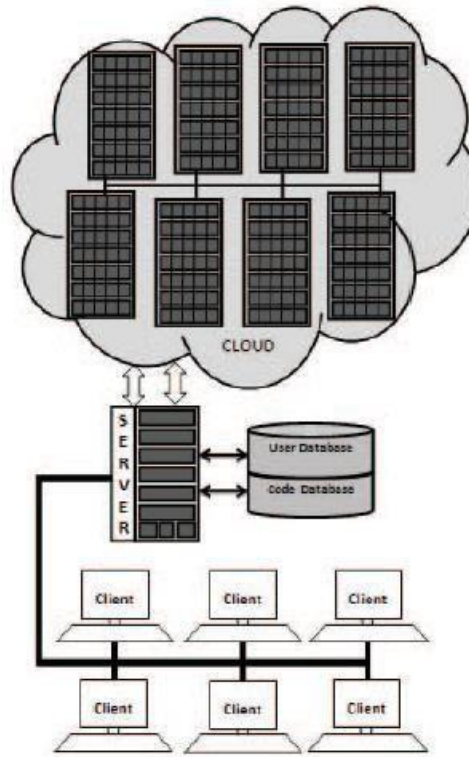
1. Compile option: This functionality allows the user to compile a program typed in the editor area. Upon clicking this option, the user's program is submitted to the cloud server. The cloud server does the compilation and returns the appropriate result.
2. Execute option: When this option is clicked, the user is provided with the links to the executable file of the program for him or her to download and subsequently execute. The user has a folder where all programs previously compiled at least once and without errors are stored.
3. Start test option: This option allows the user to start writing code. Unless this button is clicked the user cannot start writing code.

All users' programs together with the timestamps of when they were compiled are stored at the server-side database. One notable feature of this work is that users are not allowed to execute their code on the server. Instead a URL is provided for the user to download the executable file. The feature of downloading the executable file onto the user's terminal ensures that malicious code (for example code to format the C: drive on the server itself) written on the server will not execute on the server itself (thereby keeping the server intact and safe).

##### ***2.5.4.1 Project architecture***

The system uses a dual-layered architecture. The lower layer consists of clients, which are of lower configuration. The upper layer consists of the server. The important components of the upper layer are described in Figure 2.6 below:

1. A web framework, Visual Studio 2010, which handles the work of scripting and compilation of code;
2. IIS server which handles the client request;
3. Database which stores the client information; and
4. The “cloud hard disk” as a shared resource.



**Figure 2.6: Architecture of the project**

#### ***2.5.4.2 Project Implementation***

The user interface of this research was programmed in HTML and enhanced with ASPX. It was assumed that the user would use his or her favourite text editor to create and correct program files. This assumption allowed the creation of a very simple front end that loads quickly and is platform independent. Although the front end is designed to be as simple as possible with only a few commonly used options, it is sufficiently functional and can be used quickly. The server-side part of the application was implemented using ASPX written in ASP.NET that handles the communication between a user and compiler. The script does the file

managing, runs compilers and processes the compilation results. The result is the source code listing or a list of errors sent back to the user.

#### ***2.5.4.3 Limitation of the system***

The major setback of this system developed by the authors in [2] is that the system only returns the executable (.exe) of any program successfully compiled which requires the user to carry out some installation before being able to execute the executable returned.

Secondly this implementation was designed only for the C programming language and does not provide a mobile application, leaving the user with the single option of accessing the system only via a web browser.

### **2.5.5 Cloud Compiler Based on Android**

In 2014, Vijay et al. [3] developed a cloud compiler based on Android. Unlike Aamir et al., Vijay et al. developed an Android-based IDE that could detect the programming language of the code typed.

Vijay et al. identified two simple but substantial problems with today's IDEs. Firstly they require intensive CPU and memory usage which is not available all the time and since these applications are installed on a specific system, it prevents portability. By combining cloud computing and Android technologies, their project aimed to remove the requirement for powerful systems and provide portability to the developer, considering that the Android application provides much better functionality than other heavy programming kits for the Android. The compiler they built was embedded in the cloud and used SaaS. The compiler responds by providing the output of the program if successfully compiled or returns errors and warnings.

#### ***2.5.5.1 Architecture of cloud compiler based on Android***

The system was designed to work for three fields, which they named zones. So, the system was divided into three zones as shown in Figure 2.6.

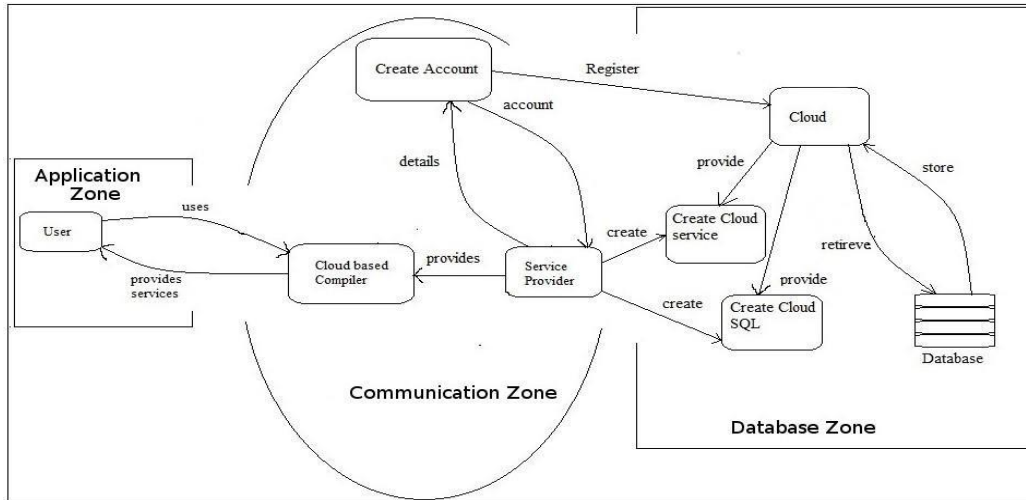


**Application zone:** The application zone consists of the interface from which a client can interact with the proposed system. The modules included in this zone are the Android application and the browser. The Android application is only for versions 2.3 (Gingerbread) and above. Any person who does not have Android can also use the proposed system through a web browser. These interfaces will provide the user with editors and various options through which user can access functions needed such as compile and file upload. The application zones must be provided with an internet connection. Without an internet connection, the compilers cannot be used. Users write and store their code using the editor provided. Whenever the device gets an internet connection, the files will be uploaded automatically if the user chooses.

**Communication zone:** When code is being sent for compilation, the flow moves into the communication zone. The communication zone is the core part of the model. First, the code's language is detected so that the code should be sent to appropriate compiler. The communication zone also includes scheduling the compilation queue and checking whether the compiler is idle or not; if it is not then the code goes to a wait state. After the compiler is detected in the idle state, the code is sent directly for execution. For getting access to the workspace, the user has to register for the first time and then log in. This transferring of user name and passwords in an encrypted format is included under the communication zone.

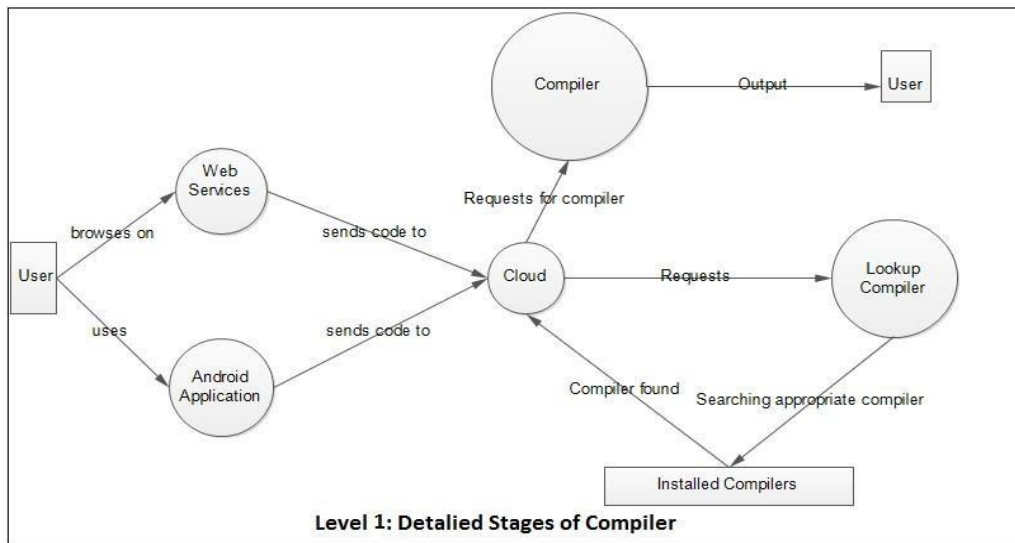
**Database zone:** The database zone consists of total back-end contents such as workspace, user name and passwords. These passwords are saved in encrypted format in the database.

The users are provided with a limited workspace for storing their codes or projects. Whenever any particular user logs in, he or she will be provided with their workspace only. These files are accessible either from the Android application or from the web browser.



**Figure 2.7: Architecture of the system**

### 2.5.5.2 Dataflow diagram of the system



**Figure 2.8: Dataflow diagram of the system**

### 2.5.5.3 Implementation of cloud compiler based on android

The mobile application was developed using Android programming. The web version of the application was developed with HTML and PHP.

### 2.5.5.4 Implementation of cloud compiler based on Android

The limitations of this system are that it does not accept runtime inputs for programs that require such interaction, and this system does not make provision for running GUI-based programs in any way.

## **2.5.6 Cloud-based “C - Programming” Android application framework**

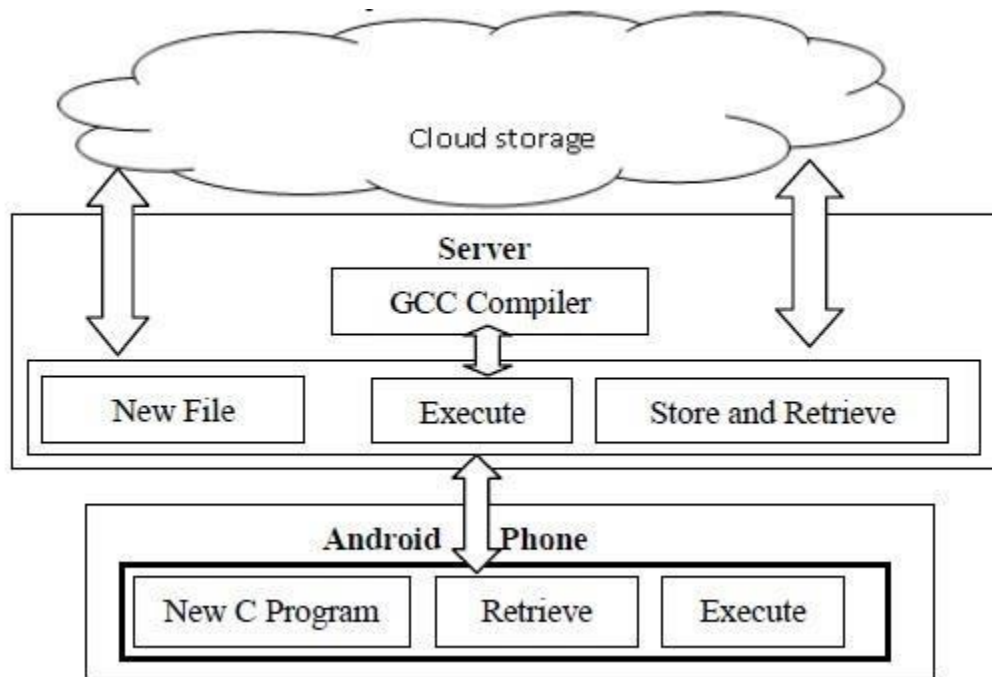
In 2015, Sonali et al. [4] developed a cloud-based “C - Programming” Android Application Framework. In their work, they built an Android-based IDE for running C programs on users’ smart devices.

### ***2.5.6.1 Functionality of the system***

For a user to use this system, he or she has to create an account. Upon doing this, the user is assigned a user ID by the system which is stored in the database. With the user ID, a user can log into the system and use the services provided. The Android application is made up of the editor, submit and save, error box and result box (output field). The C code will be edited in the editor of mobile application after which the code must be submitted to the server for further processing. The server hosted in the cloud has the GNU Compiler Collection (GCC) compiler installed on it.

Submitted code is compiled and the result is returned to the user’s mobile device. The output is displayed in the output panel for successfully compiled programs, otherwise appropriate error messages are displayed in the error panel on the user’s device. The system can also be accessed from a browser.

### 2.5.6.2 Functionality of the system



**Figure 2.9: System Architecture**

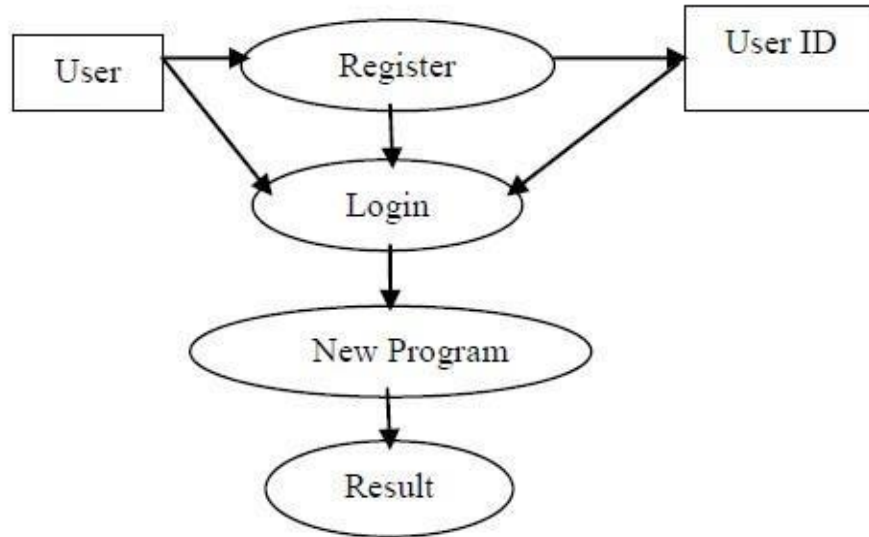
**DNS3 C PRO Cloud:** This module consists of a Java-based tool for configuring the GCC compiler. The tool can compile and execute the C programs and the results will be shown to the user as seen in Figure 2.8. It is a browser-based app that can be accessible from any browser on the network and also on any smartphone device having groups or Wi-Fi support.

**Android application:** The user interface on an Android phone in which user can type the C program is developed here. Also compile and execute functionality will be separately provided on the phone, and the result will be displayed in the result box. The phone will be connected to the cloud server via a Wi-Fi network.

**Server Application:** This service acts as an intermediate layer between a C pro application and an Android phone. Users' requests for compilation and execution of C programs are forwarded to these services and the results returned to the phone.

### 2.5.6.3 Dataflow diagram of the system

The diagram in Figure 2.10 shows the flow of data in the system.



**Figure 2.10: Data flow diagram of the system**

### 2.5.6.4 Implementation

The system was implemented with for the Android mobile operating system using the Android SDK 20 with minimum Android OS API2.2. The database system used was MySQL.

In general, it is worthwhile to note that aside from providing a programmer the opportunity to code on the go with a smart device, there are quite a number of other advantages that cloud-based compilers have to offer. Some of these advantages are:

1. User's device memory to install a compiler is saved.
2. There is no need to upgrade the compiler, as every part of maintenance is handled by the cloud provider.
3. The user will not have to install compiler on different devices, just connect to the service and use it.

#### ***2.5.6.5 Limitations of the system***

This system has no support for any other language than the C programming language.

### **2.6 Proposed solution to limitations of the existing works**

To address the limitation posed by Mohammed et al. In [1], the proposed system is built for the C/C++ programming language. Also, a hybrid mobile application framework will be used for the development in order to publish the app for both Android and iOS users.

The system developed by Vijay et al. [3] could accept runtime inputs for programs. The proposed system is to be developed to allow users to send inputs required by their programs for compilation and execution.

The work of Sonali et al. [4], has the limitation of only compiling and executing C programs. The new system will implement a C++ based compiler.

## **CHAPTER THREE RESEARCH METHODOLOGY**

This chapter will focus on what needs to be done to manage the proposed system from start to finish. It will describe every step in the project life cycle in depth so as to know exactly which tasks to complete, when and how. It will contain a body of practices, procedures, methods and phases used by anyone who will make use of the proposed system.

### **3.1 System analysis**

System analysis is a technique to help define:

- What the system needs to do (processing requirements);
- What data the system needs to store and use (data requirements);

- What inputs and outputs are needed; and
- How the functions work together to accomplish tasks

### **3.2 Analysis of the proposed system**

The proposed system will use Ionic 3.0 Framework with a native Apache Cordova wrapper, Angular 4, Typescript 2.2 and Cpp.sh online compiler. This chapter also gives a description and architecture of the proposed system.

### **3.3 Ionic framework**

Ionic framework is a front-end SDK that lets you write mobile apps in HTML, CSS and JavaScript, and then build and run them in iOS, Android and Windows phone. It is an open source project, so it is completely free to use. It provides the following components:

- CSS Components – That enable you to apply the right style to your HTML element in your application;
- JavaScript/Angular/ Typescript – Provides not only styling but behaviour, such as navigation or gestures like swiping etc.
- Icons – These are collection of icons and fonts for your mobile application.

Ionic and all these components together form a regular web application all in HTML, CSS and JavaScript at run time. But to run this as a mobile app you need a way to lunch this web application in a native WebView that is essentially a special web browser known as Cordova. Cordova builds on top of various mobile SDKs by providing plugins to access native APIs like console, camera, geolocation, etc. and command line interface (CLI) that make it easy to create projects, build run and so on. So, anything that has to do with bridging to native code is Cordova's responsibility.

### 3.4 CPP.SH online compiler

This is a simple front-end for a GCC. The system uses GCC 4.9.2 with boost C++ library version 1.55.0.

### 3.5 System requirements analysis

System requirements analysis encompasses those tasks that go into determining the needs or conditions to meet for a new or altered solution, taking account of the possibly conflicting requirements of the various stakeholders, such as users. It can also be applied specifically to the analysis proper. There are two major types of requirements, namely functional and non-functional requirements.

**Functional requirements** - This highlights a list of functions that the proposed systems must address.

- C/C++ Code Compilation: Users' ability to compile and run their programs on their smart devices.
- C/C++ Learning Components: Users' ability to find learning resources in formats such as PowerPoint slides, audio and video tutorials.

**Non-functional requirement** - The non-functional requirements for a system are typically constraints on the functional requirements – that is, not what the system does, but how it does it (e.g. how quickly, how efficiently, how easily from the user's perspective, etc.).

#### **System requirement:**

- This system will run successfully on Android 4.1 (known as Jelly Bean) and above with optimal memory performance or a PC with a browser installed.

#### **Usability requirements:**

- The users will be able to input data from the forms with 98% accuracy in a mean time of less than 10 minutes.



**Security requirements:**

- Constraints such as primary key, allow null value, triggers etc. will be used in the back end of this system to ensure data integrity and confidentiality.

**Delivery requirements:**

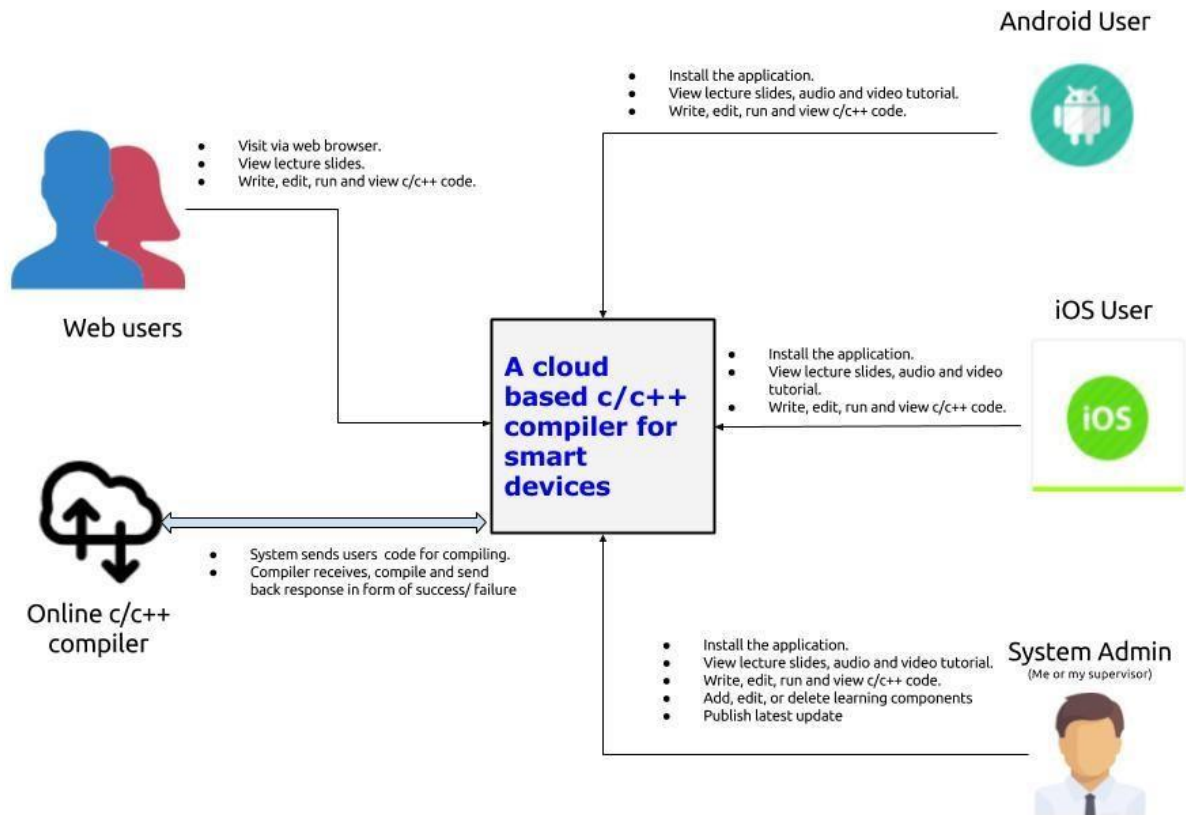
- Each life cycle phase of this system will be delivered on a monthly basis to my supervisor while the full system will be delivered by first week in November.

**3.6 Data flow diagram (DFD)**

A DFD shows the flow of the data among a set of components. The components may be tasks, software components or even abstractions of the functionality that will be included in the system.

- **The Process** - The first component of the DFD is known as a process. It shows a part of the system that transforms inputs into outputs.
- **The Flow** - This is represented graphically by an arrow into or out of a process. It is used to describe the movement of chunks, or packets of information from one part of the system to another part.
- **The Store** - The store is used to model a collection of data packets at rest. The notation for a store is two parallel lines.

### 3.7 Context level data flow diagram



**Figure 3.1: Context level data flow diagram**

Figure 3.1 above shows the interactions between a system and other actors (external factors) with which the system is designed to interface. Either the iOS or Android user can install the application on their respective device, view lecture slides, audio and video tutorial and be able to write, edit, run and view the result of the compiled code.

Web users can achieve the above only on a PC that has a web browser installed. The System Admin has the overall privileges as he or she can add, edit or delete lecture slides, audio and video tutorial and can also publish latest updates after adding a new feature. Finally, each user's compilation request is forward to the

cloud server, which interprets the result and returns with a response which is displayed to the user on the device.

### **3.8 System design**

Next to the system analysis is the system design. System design is defined as the process of defining architecture, components, interfaces and data for a system to satisfy requirements.

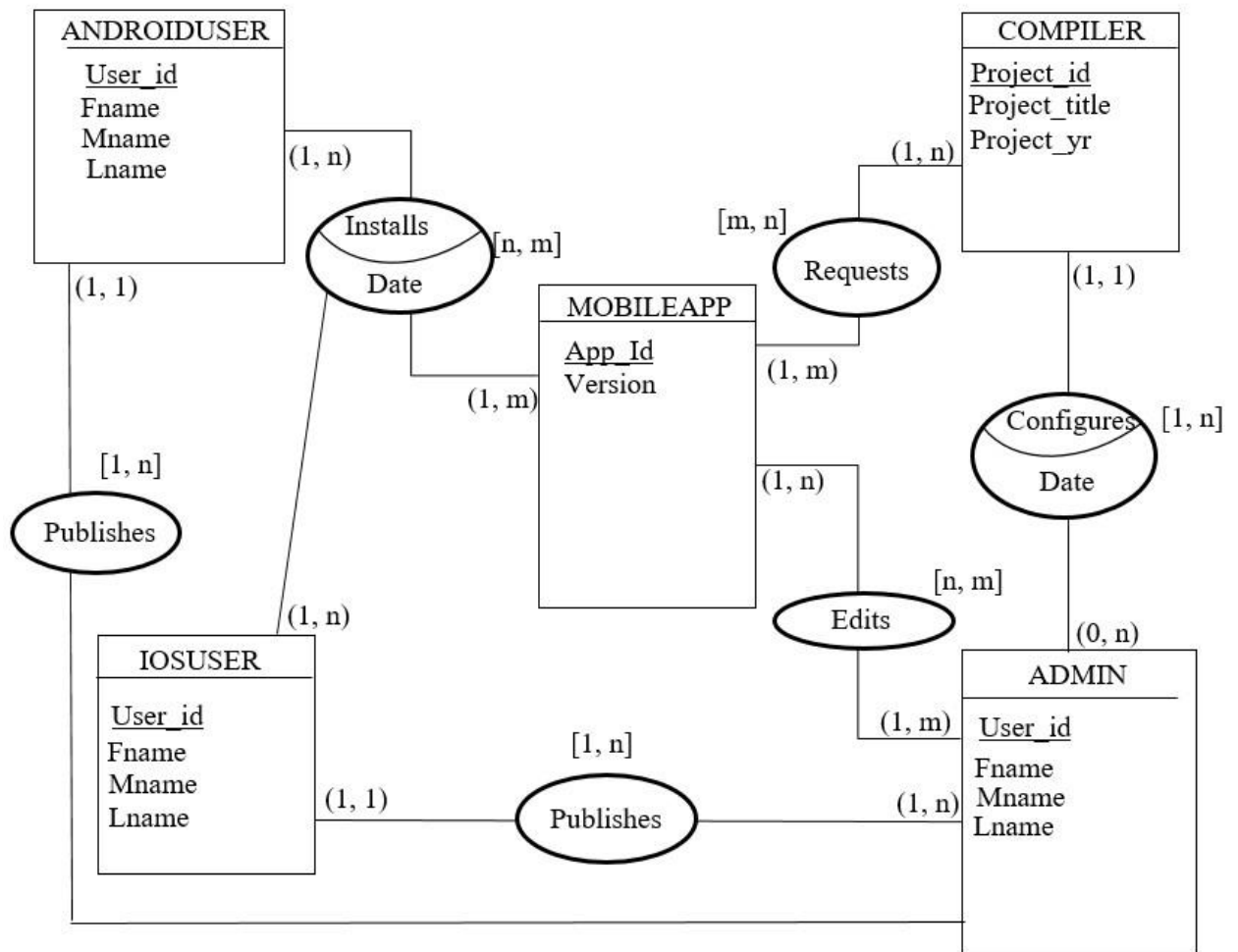
#### **3.8.1 Entity relation model**

*3.8.1.1 Entity relation model represents the real-world elements with four main concepts.*

- Entities: An entity is a person, place, object, event or concept in the user environment about which the organization wishes to maintain data.
- Attributes: Each entity type has a set of attributes associated with it. An attribute is a property or characteristic of an entity that is of interest to the system.
- Cardinality: The cardinality of one data table with respect to another data table is a critical aspect of database design. Relationships between data tables define cardinality when explaining how each table links to another.
- Relationship: A relationship is an association between the instances of one or more entity types that are of interest to the system. Relationships are labelled with verb phrases.

#### **3.8.2 Entity relationship diagram (ERD)**

ERD is a detailed, logical and graphical representation of the data for a system. A rectangle is used to represent an entity, and lines are used to represent the relationship between two or more entities. The basic entity-relationship modeling notation uses three main constructs: data entities, relationships and their associated attributes.



**Figure 3.2: Entity-relationship diagram**

The entity-relationship diagram shown in figure 3.2 has the following strong entity sets: ANDROIDUSER, IOSUSER, COMPILER, MOBILEAPP and ADMIN; the following are the relationship sets: submits, installs, publishes, edits and configures. There is a one-to-many relationship between ANDROIDUSER and ADMIN, IOSUSER and ADMIN and COMPILER and ADMIN. All other entities have a many-to-many relationship with MOBILEAPP.

### 3.8.3 Relational database model

Relational Database Model is derived from entity relation model by using the following five rules

- To each entity corresponds a relation (Entity name \_\_\_\_ relation name);
- To each attribute of an entity corresponds an attribute of the relation;
- The identifier of the entity becomes the key of the relation;
- For associations of maximum cardinality [1: n], add the key of the relation on the n side to the relation on the 1 side; and
- For associations of maximum cardinality [n: m], a new relation should be created using the concatenation of the keys of the associated relations as the key. The attributes of the association should be added as attributes of the new relation.

#### 3.8.4 Relational database model of the proposed system

The proposed system will implement the following database model:

🚩 **ANDROIDUSER** (User\_id, Fname, Mname, Lname)

🚩 **IOSUSER** (User\_id, Fname, Mname, Lname)

🚩 **MOBILEAPP** (App\_Id, Version)

🚩 **ADMIN** (User\_id, Fname, Mname, Lname)

### 3.9 Database file design

The database file design is directly derived from the relational database model by converting it to tables with the following as shown in Figures 3.3, 3.4, 3.5, 3.6 and 3.7 below.

VGG-LT-080\MSSQ...dbo.ANDROIDUSER ×			
	Column Name	Data Type	Allow Nulls
▶ 🔑	User_Id	int	<input type="checkbox"/>
	Fname	nvarchar(50)	<input type="checkbox"/>
	Mname	nvarchar(50)	<input checked="" type="checkbox"/>
	Lname	nvarchar(50)	<input type="checkbox"/>
			<input type="checkbox"/>

Figure 3.3: Android user table

VGG-LT-080\MSSQL...DB - dbo.IOSUSER × VGG-LT-080\MSSQ...dbo.ANDROIDUSER			
	Column Name	Data Type	Allow Nulls
🔑	User_Id	int	<input type="checkbox"/>
	Fname	nvarchar(50)	<input type="checkbox"/>
	Mname	nvarchar(50)	<input checked="" type="checkbox"/>
▶	Lname	nvarchar(50)	<input type="checkbox"/>
			<input type="checkbox"/>

Figure 3.4: iOS user table

VGG-LT-080\MSSQL...- dbo.MOBILEAPP ×			
	Column Name	Data Type	Allow Nulls
▶ 🔑	App_Id	int	<input type="checkbox"/>
	Version	nvarchar(50)	<input type="checkbox"/>
	PublishedDate	datetime	<input checked="" type="checkbox"/>

Figure 3.5: Mobile app table

VGG-LT-080\MSSQ...erDB - dbo.ADMIN			
	Column Name	Data Type	Allow Nulls
	User_Id	int	<input type="checkbox"/>
	Fname	nvarchar(50)	<input type="checkbox"/>
	Mname	nvarchar(50)	<input checked="" type="checkbox"/>
	Lname	nvarchar(50)	<input type="checkbox"/>

Figure 3.6: Admin Table

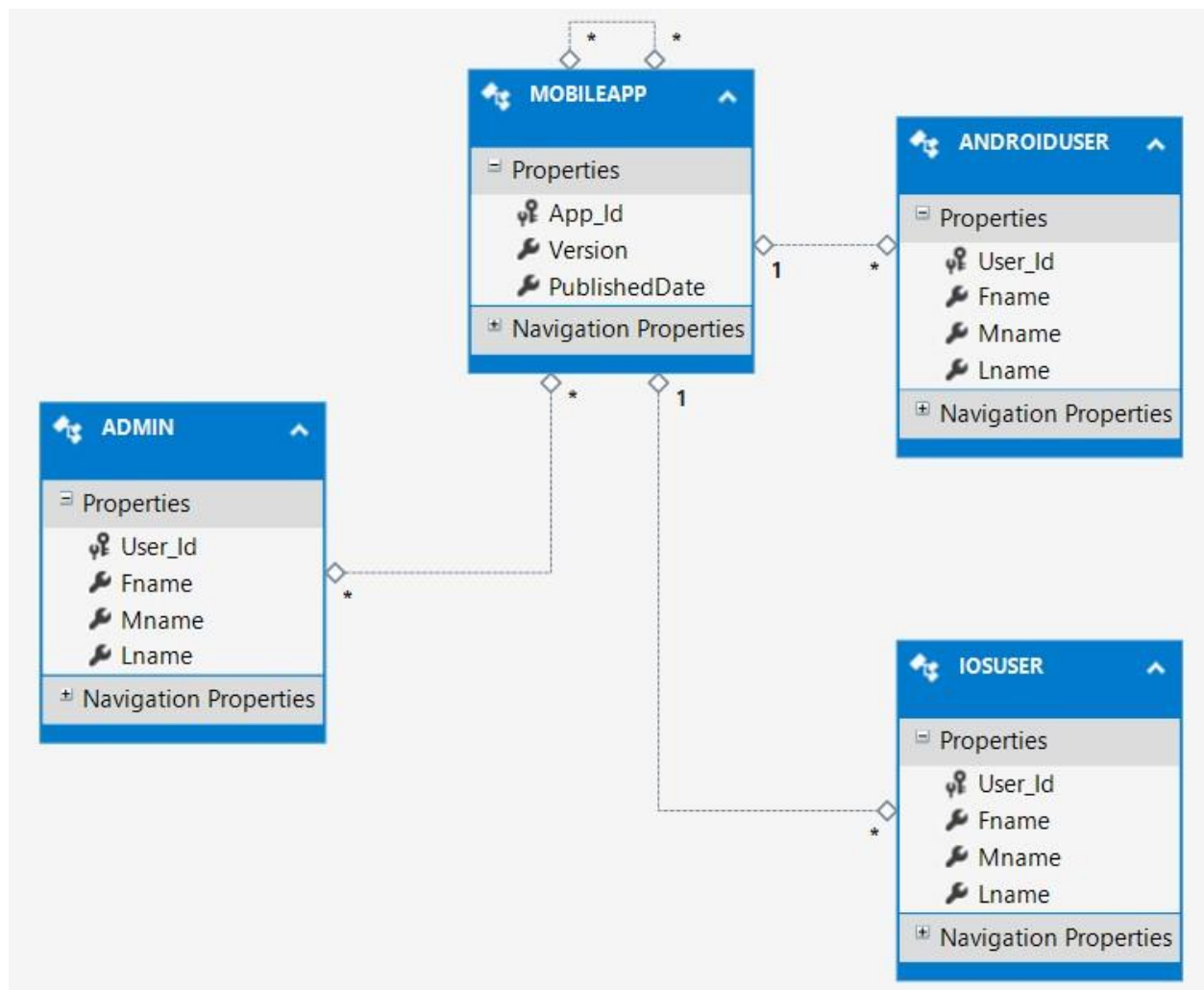


Figure 3.7: Relational database design

### 3.10 Modules of the proposed system

A module is a logical collection of functionality and resources that is packaged in a way that can be separately developed, tested, deployed and integrated into an application. Modules are independent of one another but can communicate with each other in a loosely coupled fashion. Below are the modules of the proposed system.

- **Module 1 – User Interface** – This module will focus on the general information of the system to the various users. It will include the Home page with navigation menu to all other page of the system.
- **Module 2 – Lecture Slides** – This Module will focus on gathering necessary information about C/C++ programming language in a PowerPoint slide-like manner.
- **Module 3 – Audio Lecture Module** – This will be providing C/C++ programming language learning resources in an audio manner.
- **Module 4 – Video Lecture Module** – This module will be presenting the C/C++ programming language learning resources in a video manner.
- **Module 3 – Online Compiler Module** – This module will be covering the entire process of running, debugging, compiling and displaying result of a C/C++ program.

## CHAPTER FOUR IMPLEMENTATION OF THE SYSTEM

This chapter illustrates the implementation of the designed system. The description of tools used in the design of the system and the test measures carried out on the system are also explained.

### 4.1 Tools used

In developing the system, we used the following tools.



- **Visual Studio Code:** An IDE used to develop mobile, web applications etc. with support for several languages e.g. CSS, JavaScript, Angular, C++, C# etc.
- **Server Management Studio:** A GUI tool use for script editing, query executing etc.
- **Ionic CLI:** An open source tool that helps you setup, build and run an Ionic project.
- **Git Bash:** A Git for Windows operating system which provides a BASH emulation used to run Git from the command line.

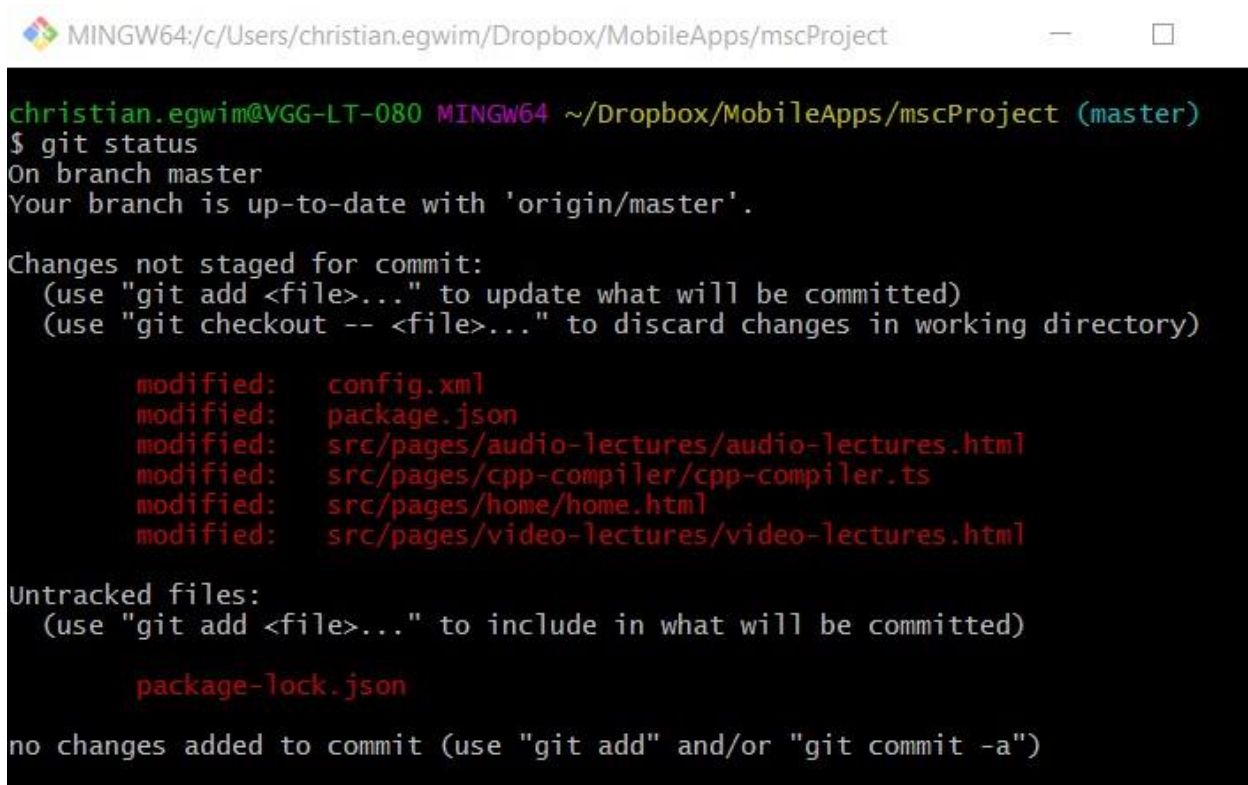
A detailed view of individual modules of the developed system is described in Figures 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8 and 4.9 below with the following:

- A screenshot of each module.
- A description of the purpose of the screen.

## 4.2 Command line interface

This is the CLI detailing the versions of Node, Cordova and Ionic installed on my Windows operating system.





```
christian.egwim@VGG-LT-080 MINGW64 ~/Dropbox/MobileApps/mscProject (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   config.xml
        modified:   package.json
        modified:   src/pages/audio-lectures/audio-lectures.html
        modified:   src/pages/cpp-compiler/cpp-compiler.ts
        modified:   src/pages/home/home.html
        modified:   src/pages/video-lectures/video-lectures.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

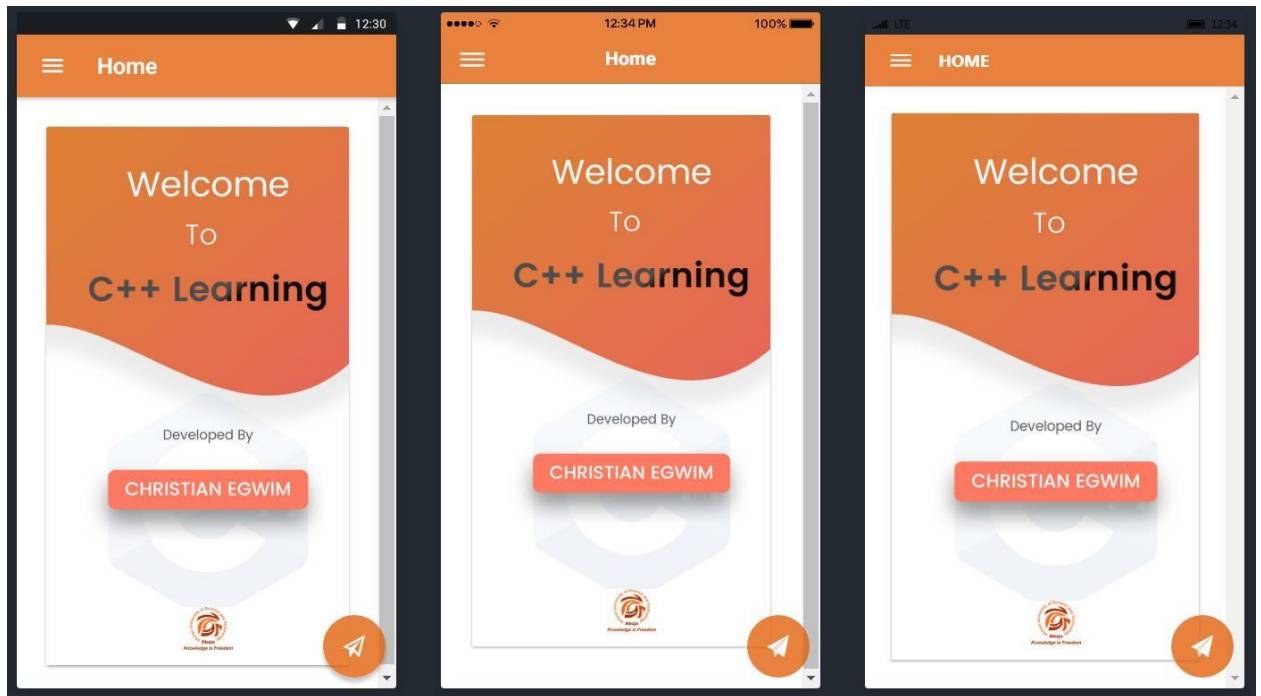
        package-lock.json

no changes added to commit (use "git add" and/or "git commit -a")
```

Figure 4.2: GIT

## 4.4 Home page

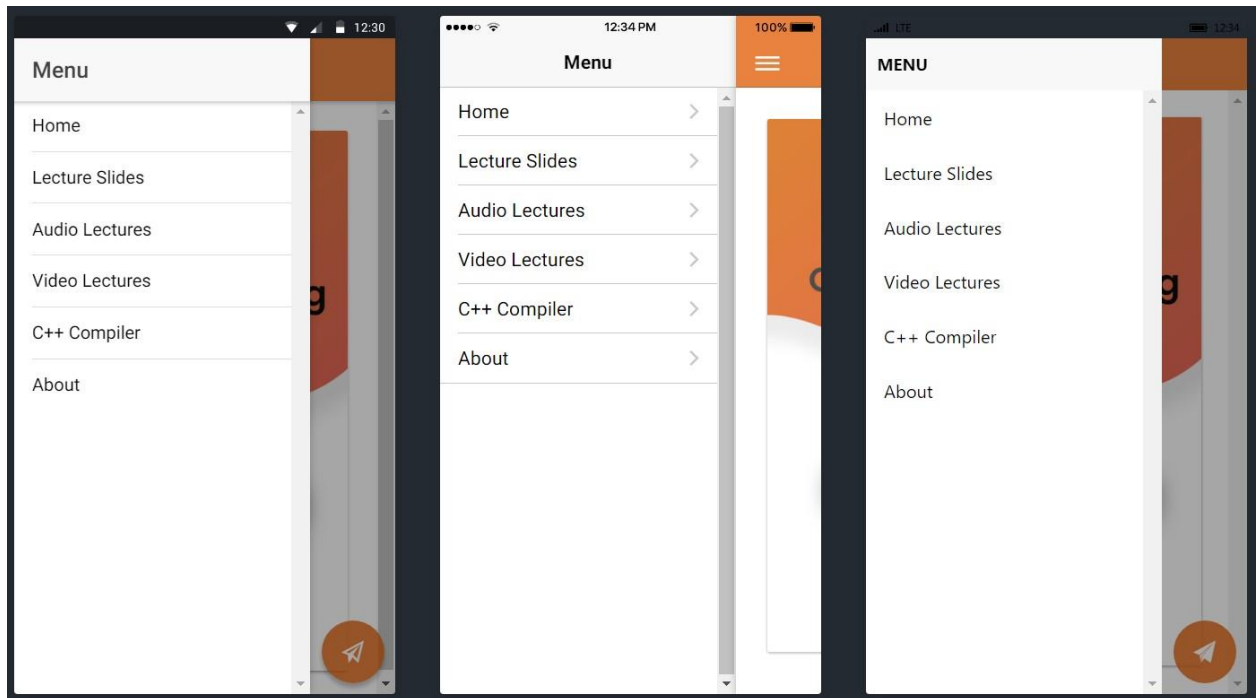
This is the landing page view starting from the left hand side of the mobile app on an Android, iOS and Windows phone emulator.



**Figure 4.3: Home page**

## 4.5 The menu bar page

This displays the list of all the menu as viewed on the different platforms. They include home, lecture slides, audio lectures, video lectures and “About”.



**Figure 4.4:** Menu bar page

The lecture slide page displays C++ language tutorial in form of slides where users can scroll through the application as shown in the respective mobile platforms below.

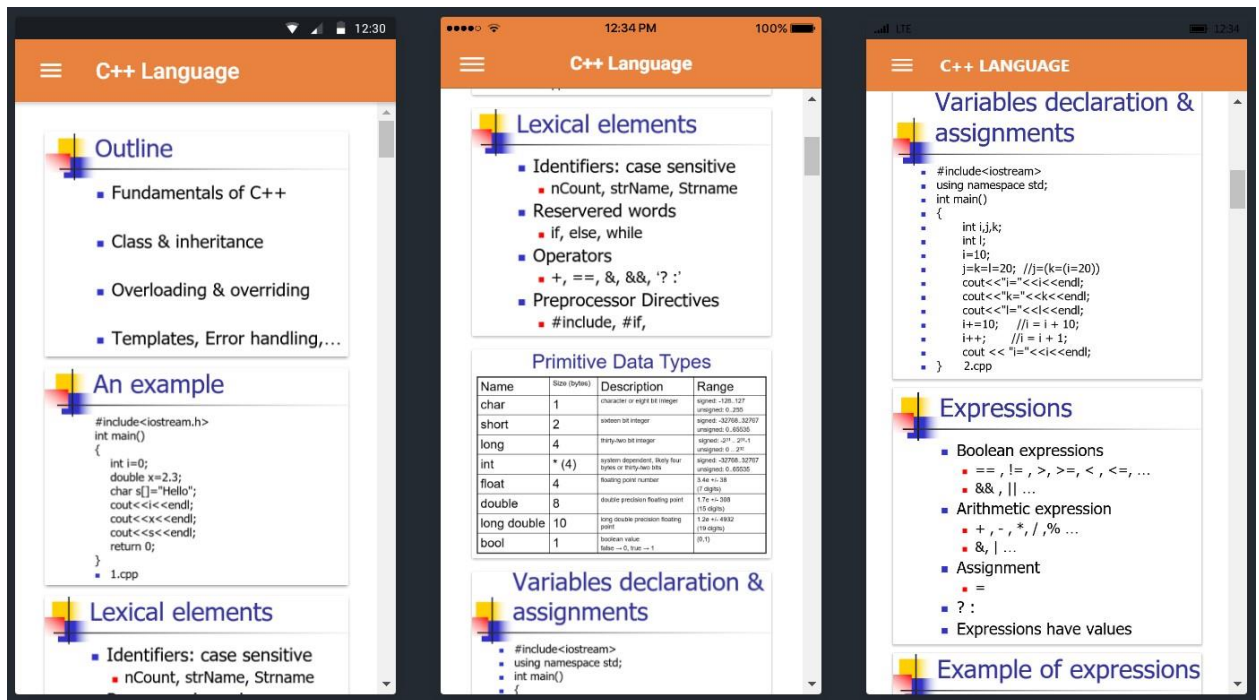
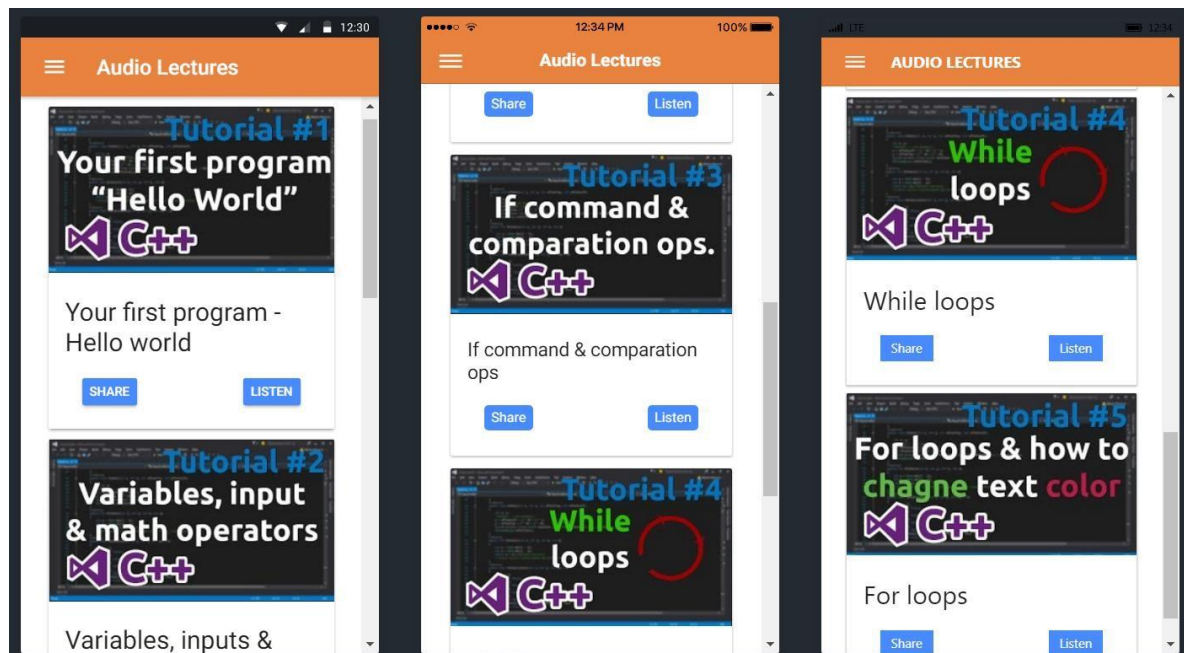


Figure 4.5: Lecture slide page

## 4.6 The audio lectures page

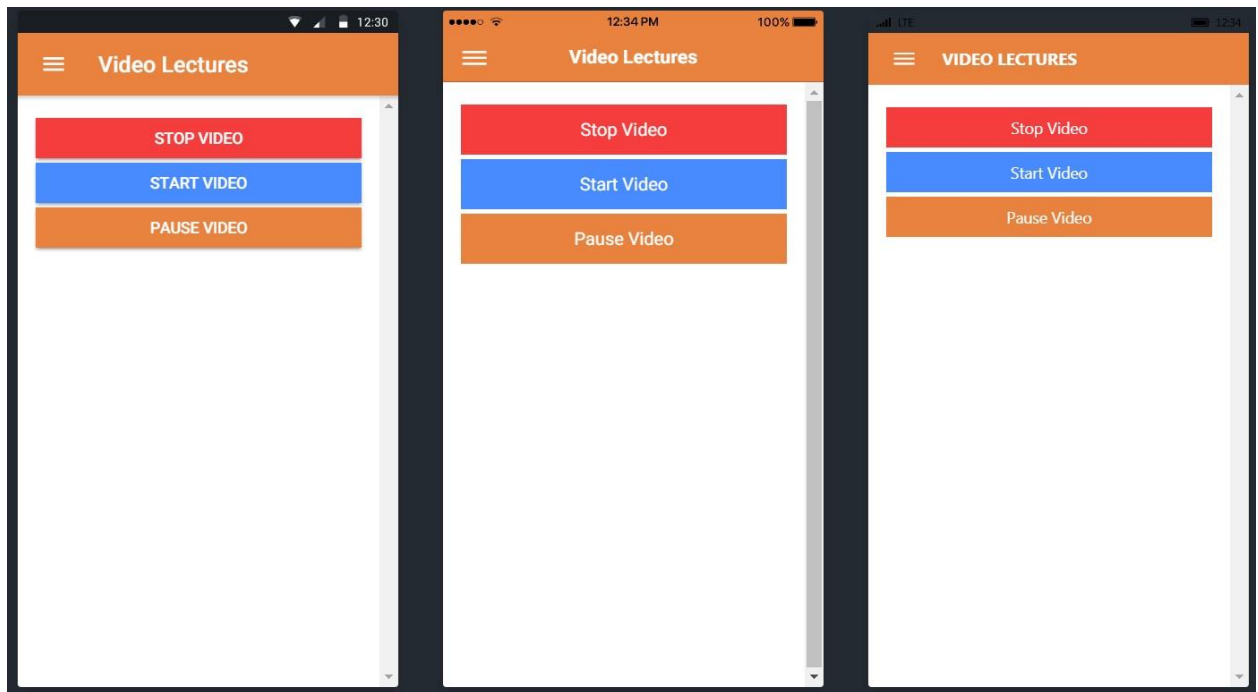
Here users on the different platforms can choose to listen to audio tutorial lectures on the C++ programming language.



**Figure 4.6: The audio lecture page**

#### **4.7 The video lectures page**

The video lecture page gives users the ability to watch a video tutorial on the C++ language on the respective platforms.



**Figure 4.7: Video lecture page**

#### **4.8 The online compiler page**

This page allows the users to write, edit and run their C++ programs on the respective platforms: The Android emulator displayed “Hi AUST students”, the iOS shows “Compiling please wait” and the Windows phone emulator displays “Running from Windows phone!”



Figure 4.8: Online compiler page



## 4.9 The project information view

This page simply gives a summary detail of research work and the research involved which basically includes my supervisor and me.

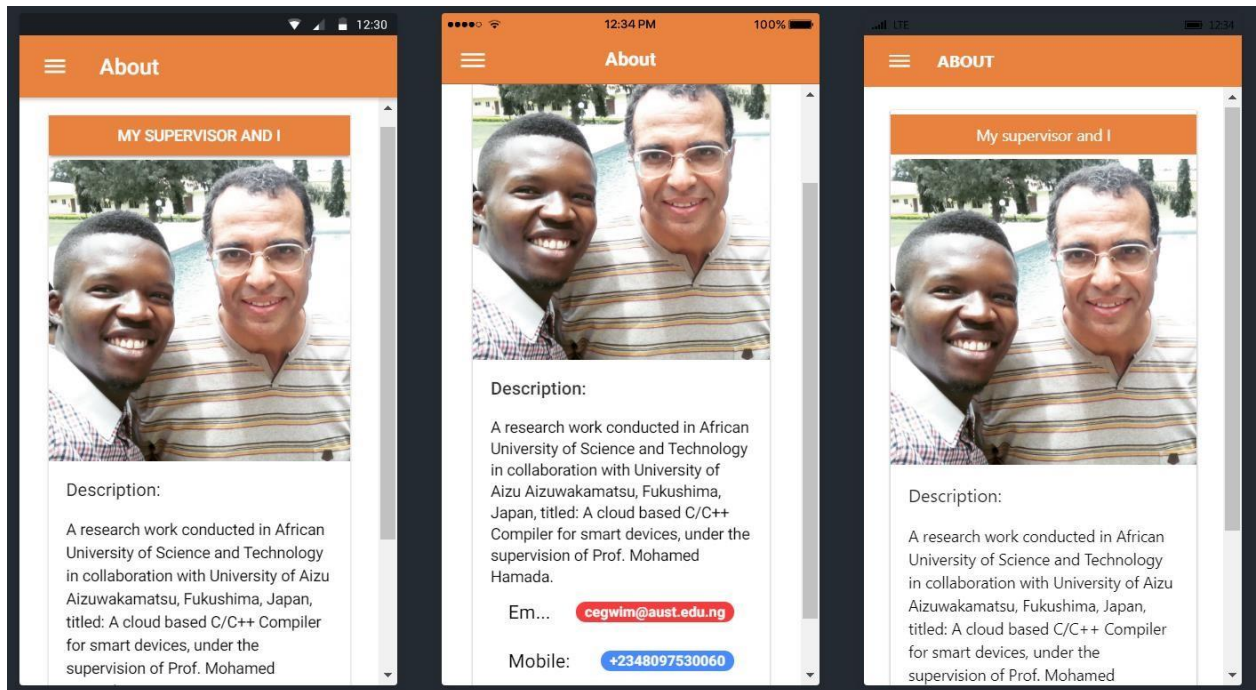


Figure 4.9: “About” page

## CHAPTER FIVE SUMMARY, CONCLUSION AND RECOMMENDATIONS

This chapter discusses the summary of results obtained, conclusions, limitations and recommendations (future works) of the “Cloud-based C/C++ Compiler for Smart Devices”.

## 5.1 Summary

The proposed system had a number of aims and objectives as discussed in Chapter One of this thesis. Among the set-out objectives and from the analysis of the implementation of the system from the previous chapter, the following is a summary of results obtained:

- Design of a mobile app for iOS, Android and Windows phone-based IDEs: These IDE were successfully developed which allow users to create and edit C/C++ programs on their smart devices.
- Design of a web-based IDE: A web-based equivalent of the mobile application was also developed to enable users who do not use mobile platform access the system from their PCs via a web browser.

## 5.2 Conclusion

In conclusion, the system developed affords a user the opportunity to write and execute C/C++ programs on a device where there is internet connectivity. This makes it possible for a programmer to easily move around with a programming kit on the go. The online compiler for smart devices can now be integrated into a smart multimedia learning system for C/C++ programming language to allow users who are learners to program on the go with their smart devices.

## 5.3 Limitations of the system

We were faced with certain challenges during this research, the most obvious of them all being the short period of time allocated. Below are the limitations of our system.

- **Limited amount of compile time:** We were unable to build our own cloud-based server for compilation. We had to use an existing online compiler (CPP.SH), which is sandboxed, so that certain system calls failed.
- **No Execution for GUI programs:** We were unable to solve the problem posed by the work of Vijay et al. [2]. The approach we initially wanted to use did not work. We discovered a new

approach at the end of the work but could not implement it due to time constraints. Hopefully we can include it in a future work.

#### **5.4 Recommendations and future work**

The time frame for this research was too short to achieve some of the desired features of the system. Initially we were supposed to implement the cloud-based server that hosts the online compiler but this could not be done; hence we used an existing online compiler. In view of this, the following are suggested for consideration in future work:

1. Implement a cloud-based server to host the C/C++ compiler: A cloud-based server should be developed in the future. As it is we used an existing C/C++ compiler, which restricted us to their functionalities.
2. Extend to include other programming languages: The implemented system was strictly for the C/C++ programming language. In future work compilers for other programming languages such as FORTRAN, C#, etc. could be developed and integrated into this system.

### **REFERENCES**

- [1] Mohamed, T.N., Hamada, M., “A cloud based Java compiler for smart devices”, from [ieeexplore.ieee.org/document/77760742](http://ieeexplore.ieee.org/document/77760742).
- [2] Aamir, N.A., Siddharth, P., Arundhati, N., Aditya, P., Venkatesh, B. 2011. “Online C/C++ Compiler Using Cloud Computing”, IEEE Spectrum. DOI 978-1-612847740/11/\$26.00
- [3] Vijay, R.S., Guruprasad, S.I., Dilip, K.J. (2014). “Cloud Compiler Based on Android”. International Journal of Science and Research (IJSR), 3(9), 2342-2346.
- [4] Sonali, S.P., Vinod, B.I. (2015). “Cloud based C - Programming Android Application Framework”. International Journal of Computer Applications (IJCA), 115(12), 20-23.

- [5] Utkrash, L. (2013, April 14). Technology and its Role in 21<sup>st</sup> Century Education. Retrieved April 1, 2016, from <http://edtechreview.in/trends-insights/insights/277role-oftechnology-in-21st-century>
- [6] Why do we need Technology Integration? (2007, November 7). Retrieved April 2, 2016, from <http://www.edutopia.org/technology-integration-guide-importance>
- [7] Mobile Learning. Retrieved April 3, 2016, from <http://library.educause.edu/topics/teaching-and-learning/mobile-learning>
- [8] Baiyun, C., Ryan, S., Luke, B., Sue, B. (2015, June 22). Students' Mobile Learning Practices in Higher Education: A Multi-Year Study. Retrieved April 3, 2016, from <http://er.educause.edu/articles/2015/6/students-mobile-learning-practices-in-higher-education-a-multi-year-study>
- [9] Berking et al., Mobile Learning Survey Report, 2013, 5.
- [10] Amit, N. (2015, July 28). Technology and the Future of Learning. Retrieved April 3, 2016 from <https://www.td.org/Publications/Blogs/Global-HRD-Blog/2015/07/Technology-and-theFuture-of-Learning>
- [11] Java (Programming Language). (2016, April 23). Retrieved April 3, 2016, from [https://simple.wikipedia.org/wiki/Java\\_\(programming\\_language\)](https://simple.wikipedia.org/wiki/Java_(programming_language)).
- [12] Andrew, W. A., (1997). Modern Compiler Implementation in C – Basic Techniques. Cambridge University Press, pp.3-6.
- [13] Smartphone OS market Share, 2015 Q2. Retrieved April 3, 2016, from <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>
- [14] Niklaus, W., (2005). Compiler Construction. Addison-Wesley, pp.6-10.
- [15] Phases of Compiler. Retrieved April 3, 2016, from <http://www.personal.kent.edu/~rmuhamma/Compilers/MyCompiler/phase.htm>

- [16] Kirk, H., Susan, L.C., Telmo, S. (2013). Cloud Essentials. John Wiley & Sons, Inc., Indianapolis, Indiana, pp.1-47.
- [17] Types of Cloud Computing. Retrieved April 6, 2016, from <http://www.thecloudtutorial.com/cloudtypes.html>.
- [18] Nisarg, G., Rahila, S. (2010). Google Android: An Emerging Software Platform For Mobile Devices, International Journal on Computer Science and Engineering (IJCSE), Special issue, ISSN: 0975-3397. 12-17.
- [19] Ideone is powered by: Sphere Engine. Retrieved April 24, 2016, from <http://www.ideone.com/sphere-engine>.
- [20] Elkstein, M. Learn REST: A Tutorial (2008, February 9). Retrieved April 26, 2016, from <http://rest.elkstein.org/2008/02/what-is-rest.html>
- [21] API Documentation. Retrieved February 22, 2016, from <http://sphereengine.com/services/compiler/docs>
- [22] Ravishanker, K. (2014, January 1). PHP CURL POST & GET Examples – Submit form using PHP CURL. Retrieved March 5, 2016, from <http://hayageek.com/phpcurl-post-get/>
- [23] Elmasri, R. (2002). Fundamentals of Database Systems. (3<sup>rd</sup> Edition), University of Texas Press, Texas. pp 165 - 180
- [24] Ian, S. (2011). Software Engineering (8<sup>th</sup> Edition), China Machine Press, China. pp.119 - 170
- [25] What is jsTree? Retrieved May 6, 2016, from <http://www.jstree.com>.
- [26] Edit Area. Retrieved May 9, 2016, from <http://www.cdolivet.com/editarea/>
- [25] Edit Area Examples. Retrieved May 9, 2016, from [http://www.cdolivet.com/editarea/editarea/exemples/exemple\\_full.html](http://www.cdolivet.com/editarea/editarea/exemples/exemple_full.html)

- [25] WebView. Retrieved May 13, 2016, from <http://developer.android.com/reference/android/webkit/WebView.html>
- [26] MySQL Workbench. Retrieved May 14, 2016, from <http://www.mysql.com/products/workbench>
- [27] Android Studio Overview. Retrieved May 14, 2016, from <http://developer.android.com/tools/studio/index.html>
- [28] Trefis T. (photographer). (2015). *Reasons Why Nokia May Be Planning to Re-enter The Smartphone Business*. [Digital image] retrieved May 26, 2016 from <http://www.forbes.com/sites/greatspeculations/2015/07/14/reasons-why-nokiamay-beplanning-to-re-enter-the-smartphone-business/#747af80f4869>.
- [29] Mayer, R.E., Moreno, R. (2002). Aids to Computer-based Multimedia Learning. In Learning and Instruction, volume 12, 107-119

## APPENDICES

### Home.html

```
<ion-header>
  <ion-navbar color="overal" >
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>Home</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <!-- <h3>Welcome</h3>

  <p>
    If you get lost, the <a href="http://ionicframework.com/docs/v2">docs</a>
    will show you the way.
  </p>

  <button ion-button secondary menuToggle>Toggle Menu</button> -->

  <!--  -->
  <ion-card *ngFor="let homePageInfo of homePageInfos">
    <img [src]="homePageInfo.image"/>
```

```

</ion-card>

<!-- dd -->

<ion-fab right bottom>

    <button ion-fab color="overal" (click)="openWebpage(url)">
        <ion-icon name="paper-plane"></ion-icon>

    </button>

</ion-fab>
Home.ts
<ion-header>
    <ion-navbar color="overal" >
        <button ion-button menuToggle>
            <ion-icon name="menu"></ion-icon>
        </button>
        <ion-title>Home</ion-title>
    </ion-navbar>
</ion-header>

<ion-content padding>
    <!-- <h3>Welcome</h3>

    <p>
        If you get lost, the <a href="http://ionicframework.com/docs/v2">docs</a>
        will show you the way.
    </p>

    <button ion-button secondary menuToggle>Toggle Menu</button> -->
    <!--  -->

    <ion-card *ngFor="let homePageInfo of homePageInfos">
        <img [src]="homePageInfo.image"/>
    </ion-card>

    <!-- dd -->

    <ion-fab right bottom>

        <button ion-fab color="overal" (click)="openWebpage(url)">
            <ion-icon name="paper-plane"></ion-icon>

        </button>

    </ion-fab>

Cpp.html
<ion-header>
    <ion-navbar color="overal">

```

```

        <button ion-button menuToggle>
            <ion-icon name="menu"></ion-icon>
        </button>
        <ion-title>C++ Compiler </ion-title>
    </ion-navbar>
</ion-header>

<ion-content padding>
    <!-- <a href="http://cpp.sh/2dd">Click</a> -->

    <!-- <a href="#"onclick="window.open('http://cpp.sh/2dd', '_system',
    'location=yes');" >Google</a> -->

    <!-- <ion-item>
        <ion-label floating>URL</ion-label>
        <ion-input type="url" [(ngModel)]="url"></ion-input>
    </ion-item> -->

    <button ion-button block clear (click)="openWebpage(url)">Lunch
        Compiler</button>
</ion-content>
Cpp.ts
import { Component } from '@angular/core';
import { NavController, NavParams } from 'ionic-angular'; import {
InAppBrowser, InAppBrowserOptions } from '@ionic-native/in-appbrowser';

@Component({
    selector: 'page-cpp-compiler',
    templateUrl: 'cpp-compiler.html',
})
export class CppCompilerPage {

    //url: string;

    public url: string = "http://cpp.sh/2dd";

    constructor(
        private inAppBrowser: InAppBrowser,
        public navCtrl: NavController,        public
        NavParams: NavParams) {
    } //dd
    ionViewDidLoad() {
        console.log('ionViewDidLoad CppCompilerPage');
    }
}

```



```

    openWebpage(url: string) {      const
options: InAppBrowserOptions = {
zoom: 'no'      }
    const browser = this.inAppBrowser.create(url, '_self', options );
    }
}

```

### Audio.html

```

<ion-header>
  <ion-navbar color="overall">
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>Audio Lectures </ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>

  <ion-refresher (ionRefresh)="addOneSong($event)">
    <ion-refresher-content></ion-refresher-content>
  </ion-refresher>

  <ion-card *ngFor="let music of allMusic">
    <!-- <ion-card-header>
      <button (click)="addToFavourites(music)" ion-button block>
        Add to Favorites
      </button>
    </ion-card-header> -->
  <!-- ddd -->
    <img [src]="music.image">
    <ion-card-content>
      <ion-card-title>{{music.name}} </ion-card-title>

      <ion-item>
        <button ion-button item-left (click)="shareSong(music)">
          Share
        </button>
        <button ion-button item-right (click)="goToMusicPlayer(music)">
          Listen
        </button>
      </ion-item>
    </ion-card-content>
  </ion-card>

</ion-content>

```

### Audio.ts

```

import { Component } from '@angular/core';

```

```

import { NavController, NavParams, LoadingController, ActionSheetController }
  from 'ionic-angular';
import { SocialSharing } from "@ionic-native/social-sharing"; import {
AudioLecturesPlayerPage } from "../audio-lectures-player/audiolectures-
player"
import { CloudServiceProvider } from "../../providers/cloud-
services/cloudservices";

@Component({
  selector: 'page-audio-lectures',
  templateUrl: 'audio-lectures.html',
})
export class AudioLecturesPage {
  public allMusic = [];

  constructor(
    private socialSharing: SocialSharing,
    private actionSheetController: ActionSheetController,
    private loadingController: LoadingController,      public
navCtrl: NavController,
    public cloudServiceProvider: CloudServiceProvider,
    public NavParams: NavParams) {
  }

  ionViewDidLoad() {
    let allMusicLoadingController = this.loadingController.create({
content: "Loading Audio Lectures "
    });
    allMusicLoadingController.present();
    this.cloudServiceProvider.getAllAudioLectures()
      .subscribe((musicList) => {
allMusicLoadingController.dismiss();      this.allMusic
= musicList
    });
  }

  addOneSong(refresher) {
    this.cloudServiceProvider.getAllAudioLectures()
      .subscribe((oneSong) => {
this.allMusic.unshift(oneSong[0]);      refresher.complete();
    });
  }

  shareSong(music) {
    let shareSongActionSheet = this.actionSheetController.create({
title: "Share Song with Friends",      buttons: [      {
      text: "Share On Facebook",
      icon: "logo-facebook",      handler:
      () => {
        this.socialSharing.shareViaFacebook(music.name, music.image,
music.link);
      }
    }
  ]
});
  }
}

```

```

        }
    },
    {
        text: "Share On Twitter",
        icon: "logo-twitter",
        handler: () => {
            this.socialSharing.shareViaTwitter(music.name, music.image,
            music.link);
        }
    },
    {
        text: "Share",
        icon: "share",
        handler: () => {
            this.socialSharing.share(music.name, "", music.image,
            music.link);
        }
    },
    {
        text: "Cancel",
        role: "destructive"
    }
]
});
shareSongActionSheet.present();
}
goToMusicPlayer(music) {
    this.navCtrl.push(AudioLecturesPlayerPage, {
music: music
    });
}

addToFavourites(music) {
    this.cloudServiceProvider.addToFavourites(music);
}

}

```

### Video.html

```

<ion-header>
  <ion-navbar color="overal">
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>Video Lectures </ion-title>
  </ion-navbar>
</ion-header>

```

```

<ion-content padding>
  <!-- dd -->

```

```

<!-- <button ion-button (click)="streamVideo()">Play Video</button> -->

<button ion-button full color="danger">Stop Video</button>
<button ion-button full (click)="streamVideo()">Start Video</button>

<button ion-button full color="overal">Pause Video</button>

</ion-content>
Video.ts
import { Component } from '@angular/core';
import { NavController, NavParams } from 'ionic-angular';

import { VideoPlayer, VideoOptions } from '@ionic-native/video-player'; import
{ StreamingMedia, StreamingVideoOptions } from '@ionicnative/streaming-
media';

@Component({
  selector: 'page-video-lectures',
  templateUrl: 'video-lectures.html',
})
export class VideoLecturesPage {
  videoOptions:
VideoOptions;  videoUrl:
string;

  constructor(
    private streamingMedia: StreamingMedia,
    private videoPlayer: VideoPlayer,
    public navCtrl: NavController,      public
navParams: NavParams) {
  }
  ionViewDidLoad() {
    console.log('ionViewDidLoad VideoLecturesPage');
  }

  async playVideo() {
try {
  this.videoOptions = {
volume: 0.7
  }
  this.videoUrl = "http://techslides.com/demos/sample-videos/small.mp4";
await this.videoPlayer.play(this.videoUrl, this.videoOptions);
console.log("video has complited.");
}
catch(e) {
  console.error(e);
}
}
}

```

```

streamVideo() {
  let options: StreamingVideoOptions = {
    successCallback: () => { console.log('Video played') },
    errorCallback: (e) => { console.log('Error streaming') },
    orientation: 'landscape'
  };

  this.streamingMedia.playVideo('http://techslides.com/demos/samplevideos/small.mp4', options);
}
}

```

### Cloudservice.ts

```

import { Injectable } from '@angular/core';
import { Http } from '@angular/http'; import
'rxjs/add/operator/map';

/*
  Generated class for the CloudServicesProvider provider.

  See https://angular.io/guide/dependency-injection for more info on
  providers
  and Angular DI.
*/
const API: string =
'https://orangevalleycaa.org/api/music';

@Injectable()
export class CloudServicesProvider {
  public favoriteSongs = [];

  constructor(public http: Http) {
    console.log('Hello CloudServicesProvider Provider');
  }

  getMusic(){
    return this.http.get(API)
      .map(response => response.json());
  }
  getHomePageLandingImage(){
    return this.http.get('/assets/homePage.json')
      .map( response => response.json());
  }
  getMeAndProf(){
    return this.http.get('/assets/meAndProf.json')
      .map( response => response.json());
  }
  getAllAudioLectures(){
    return this.http.get('/assets/audioLectures.json')

```

```

        .map( response => response.json());
    }
    getLectureSlides() {
        return this.http.get('/assets/cppSlides.json')
            .map( response => response.json());
    }
    getOneSong() {
        let oneSongUrl = API + "/gty/1";
        return this.http.get(oneSongUrl)
            .map(response => response.json());
    }
    getFavorites() {
        return this.favoriteSongs;
    }
    addToFavourites(song) {
        let isSongAdded = this.favoriteSongs.findIndex((favoriteSong) => {
            return song.id === favoriteSong.id
        });
        if (isSongAdded === -1)
        {
            this.favoriteSongs.push(song);
        }
    }
}

```