Table of Contents

Introduction v

Problem vi

Solution Overview vii

1 Chapter 1 Embedded Systems 1

1.1. STM32F103C8 ARM based microcontroller overview 3

1.1.1. Microcontroller Overview 4

1.2. Keil - IDE Setup 7

1.2.1. Installing prerequisite STM32 Keil software pack 8

1.2.2. Create a new project. 9

1.3. Bootloader 16

1.3.1. Microcontroller Programming 16

1.3.2. Bootloader Overview 16

1.3.3. Bootloader Types 19

1.3.4. UART Bootloader 19

1.3.5. OTA Update 19

1.4. Schematic & PCB design 22

1.4.1. Touch Screen LCD 23

1.4.2. TFT Screen 24

1.4.3. RFID 25

1.4.4. NodeMCU WiFi module – ESP8266 25

1.4.5. Temperature Sensor 26

1.4.6. SPO2 Sensor 26

1.4.7. Conclusion 27

1.5. Microcontroller Drivers 27

1.5.1. RCC 27

1.5.2. GPIO 27

1.5.3. ADC 28

1.5.4. IWDG 29

1.5.5. USART 29

1.5.6. SPI 31

1.5.7. I2C 32

1.5.8. SysTick 33
1.5.9. Timer 34
1.5.10. NVIC 34
1.5.11. CRC32 35
1.6. Registration Device 36
1.6.1. System Requirements 37
1.6.2. Physical Layout 37
1.6.3. Software Architecture 38
1.6.4. Device Peripherals 41
1.7. Monitoring device 44
1.7.1. System Requirements 45
1.7.2. Physical Layout 45
1.7.3. Software Architecture 46
1.7.4. Device Peripherals 50
1.7.5. RFID reader 50
1.7.6. SPO2 driver 53
1.7.7. Temp sensor 56
1.7.8. Gateway ESP8266 56
1.7.9. Real Time Operating system 58
1.7.10. OS tasks 60
1.7.11. Monitoring Application 68
2 Chapter 2 Cloud Solution 71
2.1 Introduction 73
2.2 Cloud 73
2.2.1 What is the cloud? 73
2.2.2 How does cloud computing work? 74
2.2.3 What are the main service models of cloud computing? 75
2.2.4 Cloud Types 76
2.3 Components 77
2.3.1 Terraform 77
2.3.2 Ansible 93
2.3.3 Docker 99
2.3.4 Kubernetes 101

2.3.5 OpenStack 106
3 Chapter 3 Application 141
3.1 Web Backend 143
3.1.1 Database Design 143
3.1.2 Tools 150
3.1.3 system features 153
3.2 Mobile Application 166
3.2.1 Project Structure & Architecture: 166
3.2.2 Tools and packages 170
3.2.3 Features 170
3.3 Front End 178
3.3.1 Introduction 178
3.3.2 ALL Used Tools 178
4 Chapter 4 Data Analysis 187
4.1 Data Analysis Process 189
4.1.1 Introduction 189
4.1.2 Posing questions 190
4.1.3 Data Collection 191
4.1.4 Data wrangling 191
4.1.5 Exploratory data analysis 191
4.1.6 Communicating findings 192
4.2 Anaconda and Jupyter notebook 192
4.2.1 Anaconda 192
4.2.2 Anaconda Navigator 193
4.2.3 Conda: 193
4.2.4 Jupyter notebook 194
4.3 Python 195
4.3.1 NumPy 195
4.3.2 Pandas 196
4.3.3 Matplotlib 196
4.3.4 Seaborn 197
4.4 Data visuals 198
4.4.1 Histogram 198

4.4.2 Bar chart 198
4.4.3 Pie chart 199
4.4.4 Line chart 199
4.4.5 Box plot 200
4.4.6 Violin plot 200
4.4.7 Heatmap 201
4.4.8 Scatter plot 201
4.4.9 Faceted visuals 202
4.5 CloCare patients' data analysis 202
4.5.1 Patients' data 203
4.5.2 Dataset 203
4.5.3 Wrangling process 204
4.5.4 Analytics 205
Conclusion 221
Future Work 223
Project Team Members 224
References 225

Introduction

We are developing an embedded system with relatively low budget to be included in all hospitals which contains the necessary information about the patient, and it's stored over the cloud. The patient needs only an RFID card to get identified. The prototype for this project contains 2 devices. First device is used to retrieve patient's information or register new record. The second device is a monitoring device with capabilities to connect to the system using RFID and Wi-Fi to log the data immediately to the patient's record. The system also includes a mobile application for doctors to get alerts about patients assigned to monitor devices and to file manual reports about a case. All the data is loaded on a local server inside the hospital and uploaded to the cloud in a certain time of the day.

Problem

Hospitals and clinics in Egypt nowadays are mostly based on paperwork and are standalone organizations. This arises problems as that we don't have anyway of finding out about the patient's medical history, slow processing, and may even lead to serious medical condition due to the lack of information and medication conflict. The objective of this project -as the name suggests- is to digitalize the process using a low cost IoT based embedded system that connects and unifies the medical system all over the country and provides essential information about a patient once in a hospital or a clinic.

Solution Overview

There are several (EHR) Electronic Health Record software companies nowadays for smart hospitals locally and internationally.

In Egypt:

- Cloudpital company and Sunwave company.
- The Top EHR Vendors around the world in 2020:
- Meditech, Allscripts, Athenahealth, Cerner Corporation,
- eClinicalWorks, Epic Systems, GE Healthcare, Greenway Health.

Our solution presents a low budget device to be installed in any healthcare facility and only requires a RFID card with the patient. No extra cost nor system overhead. It also aims to be a unified database, so it can be accessed almost anywhere. It will provide real-time data and analysis over the cloud.

Socially, our solution would help small to large healthcare organizations manage patients' records and automate clinical workflow. This will result in saving cost and time for medical organizations and help them to improve quality of medical service. It will facilitate doctors' work as they can monitor the current state of the patient as well as instant access to medical history. It will help nursing staff to efficiently do their work and provide a better medical service for patients. It adds value to patients as they save their medical history without worries about data loss and provides portability of data anywhere and anytime.

Commercially, producing and selling the product should be profitable and open more working opportunities in the field of IT, embedded systems, cloud computing, networking as well as managing, marketing, accounting, and sales positions.

Environmentally, it will reduce the use of paperwork and save more trees achieving sustainable development, with clean and friendly solutions.

In our prototype, the system has 3 main components:

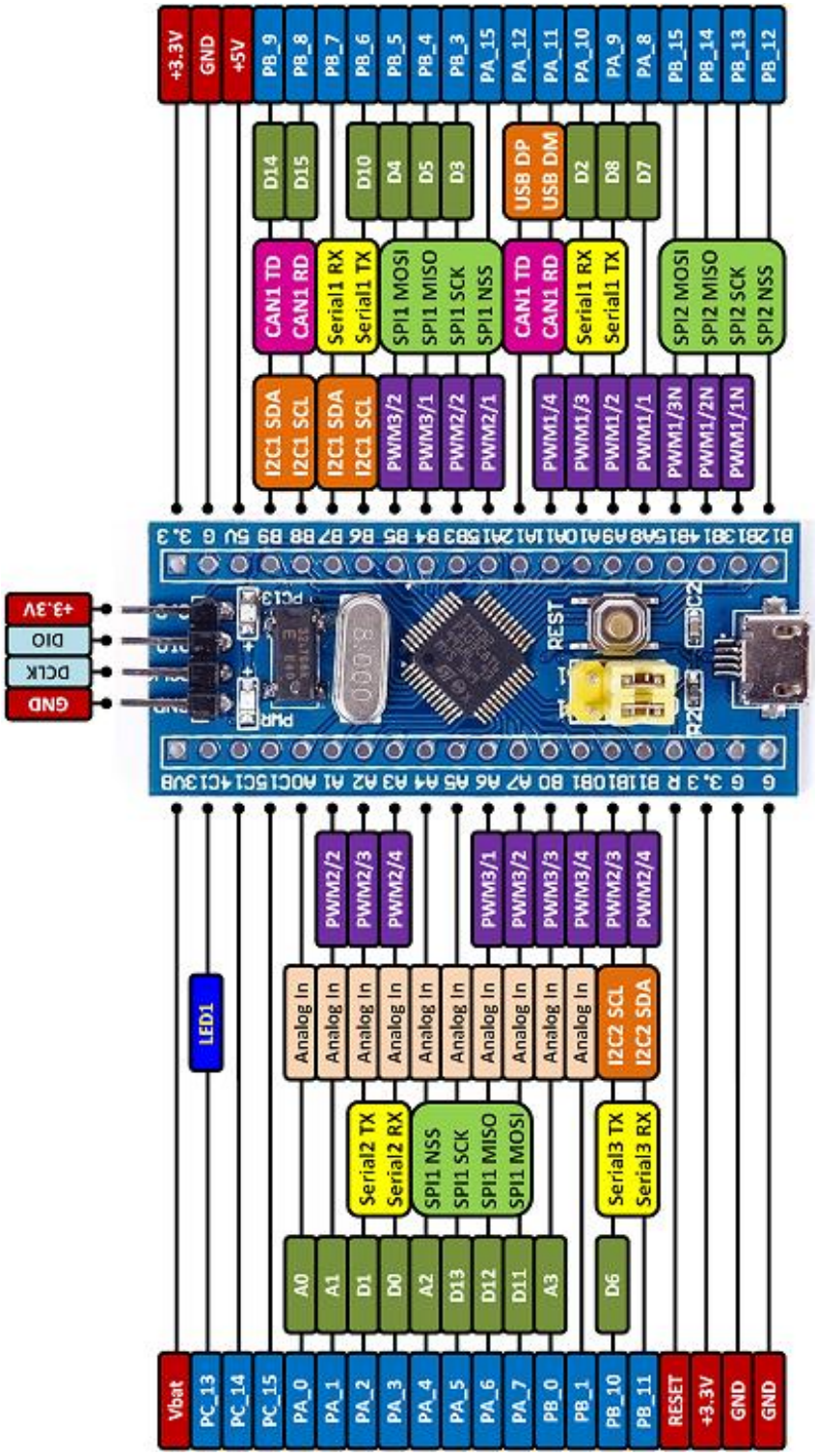
- **Mobile app:** one with the medical staff -doctors and nurses with different privileges- To get medical history from the cloud end, insert new data, get real-time updates from monitoring device, and notifications for emergencies. Patient's can be assigned using their RFID card.
- **Monitoring device:** In the prototype we'll build this device from scratch, but later in the deployment phase, we should be able to use the monitoring devices available already in hospitals to log sensors' readings directly to the patient's record. The device would recognize the patient using his RFID card.
- **Register/Log in device:** it should be placed in the reception. If the patient doesn't have RFID card or doesn't have a record in the system, the receptionist should be able to add a new record for him, and/or assign a card for him to use during his time in the hospital, that later will be used to interact with doctors and different devices in the hospital.

All components are connected to the cloud and data is centralized on the cloud.

Chapter 1

Embedded Systems

1.1. STM32F103C8 ARM based microcontroller overview



1.1.1. Microcontroller Overview

The STM32F103xx medium-density performance line family incorporates the high-performance ARM Cortex-M3 32-bit RISC core operating at a 72 MHz frequency, high-speed embedded memories (Flash memory up to 128 Kbytes and SRAM up to 20 Kbytes), and an extensive range of enhanced I/Os and peripherals connected to two APB buses. All devices offer two 12-bit ADCs, three general purpose 16-bit timers plus one PWM timer, as well as standard and advanced communication interfaces: up to two I2Cs and SPIs, three USARTs, an USB and a CAN. The devices operate from a 2.0 to 3.6 V power supply. They are available in both the –40 to +85 °C temperature range and the –40 to +105 °C extended temperature range. A comprehensive set of power-saving mode allows the design of low-power applications.

The STM32F103xx medium-density performance line family includes devices in six different package types: from 36 pins to 100 pins. Depending on the device chosen, different sets of peripherals are included.

These features make the STM32F103xx medium-density performance line microcontroller family suitable for a wide range of applications such as motor drives, application control, medical and handheld equipment, PC and gaming peripherals, GPS platforms, industrial applications, PLCs, inverters, printers, scanners, alarm systems, video intercoms, and HVACs.

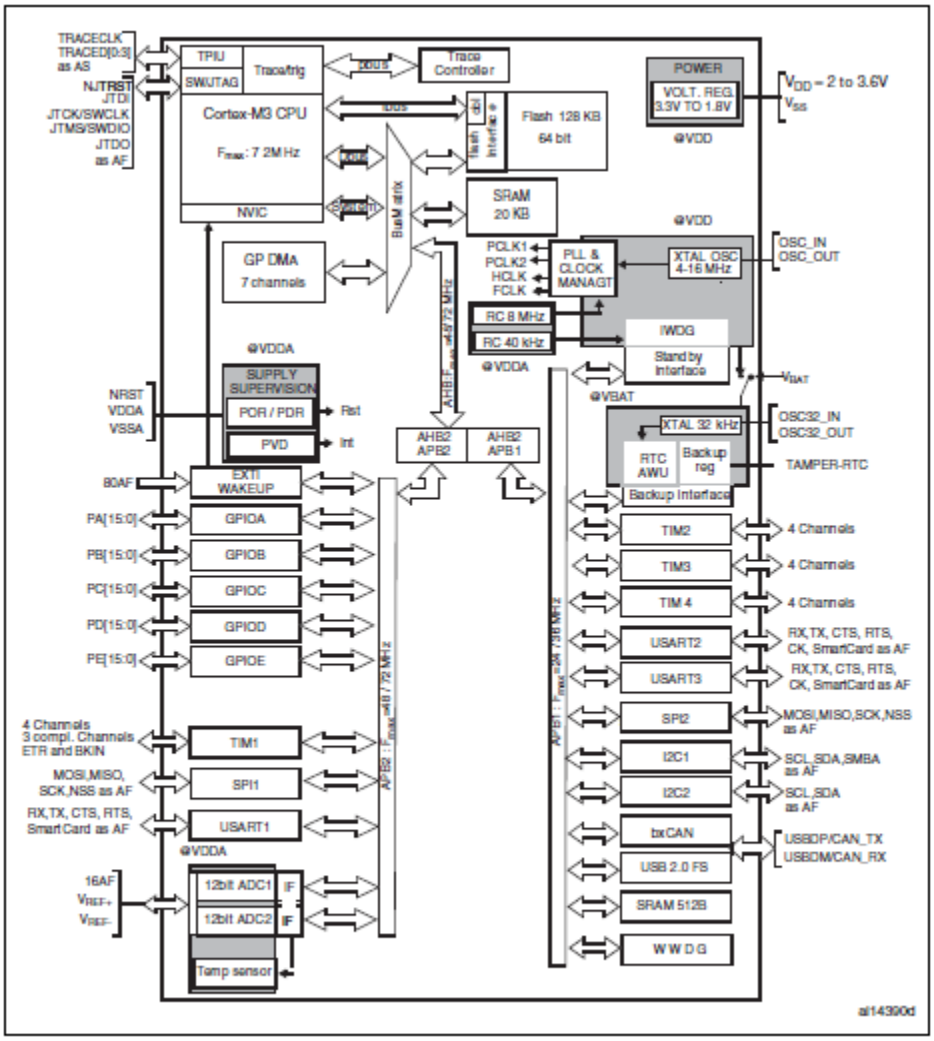
The STM32F103C8 has 72MHz clock, 64kB Flash, 20kB SRAM, PLL, Embedded Internal RC 8MHz and 32kHz, Real-Time Clock, Nested Interrupt Controller, Power Saving Modes, JTAG and SWD, 3 Synch. 16-bit Timers with Input Capture, Output Compare and PWM, 16-bit 6-ch Advanced Timer, 2 16-bit Watchdog Timers, SysTick Timer, 2 SPI, 2 I2C, 3 USART, USB 2.0 Full Speed Interface, CAN 2.0B Active, 2 12-bit 10-ch A/D Converter, Fast I/O Ports.

1.1.1.1. ARM® Cortex™-M3 core with embedded Flash and SRAM

The ARM Cortex™-M3 processor is the latest generation of ARM processors for embedded systems. It has been developed to provide a low-cost platform that meets the needs of MCU implementation, with a reduced pin count and low-power consumption, while delivering outstanding computational performance and an advanced system response to interrupts. The ARM Cortex™-M3 32-bit RISC processor features exceptional code-efficiency, delivering the high-performance expected from an ARM core in the memory size usually associated with 8- and 16-bit devices. The STM32F103xx performance line family having an embedded ARM core, is therefore compatible with all ARM tools and software.

1.1.1.2. Serial wire JTAG debug port (SWJ-DP)

The ARM SWJ-DP Interface is embedded. and is a combined JTAG and serial wire debug port that enables either a serial wire debug or a JTAG probe to be connected to the target. The JTAG TMS and TCK pins are shared with SWDIO and SWCLK, respectively, and a specific sequence on the TMS pin is used to switch between JTAG-DP and SW-DP.



1. T_A = –40 °C to +105 °C (junction temperature up to 125 °C).
2. AF = alternate function on I/O port pin.

1.1.1.3. Nested vectored interrupt controller (NVIC)

The STM32F103xx performance line embeds a nested vectored interrupt controller able to handle up to 43 maskable interrupt channels (not including the 16 interrupt lines of Cortex™-M3) and 16 priority levels.

- Closely coupled NVIC gives low-latency interrupt processing.
- Interrupt entry vector table address passed directly to the core.
- Closely coupled NVIC core interface.

- Allows early processing of interrupts.
- Processing of late arriving higher priority interrupts.
- Support for tail-chaining.
- Processor state automatically saved.
- Interrupt entry restored on interrupt exit with no instruction overhead.

1.1.1.4. SysTick timer

This timer is dedicated for OS but could also be used as a standard down counter. It
features:

- A 24-bit down counter.
- Auto reload capability.
- Maskable system interrupt generation when the counter reaches 0.
- Programmable clock source.

1.1.1.5. I²C bus

Up to two I²C bus interfaces can operate in multi master and slave modes. They can support standard and fast modes. They support dual slave addressing (7-bit only) and both 7/10-bit addressing in master mode. A hardware CRC generation / verification is embedded. They can be served by DMA and they support SM Bus 2.0/PM Bus.

1.1.1.6. Universal synchronous/asynchronous receiver transmitter (USART)

One of the USART interfaces can communicate at speeds of up to 4.5 Mbit/s. The other available interfaces communicate at up to 2.25 Mbit/s. They provide hardware management of the CTS and RTS signals, IrDA SIR ENDEC support, are ISO 7816 compliant and have LIN Master/Slave capability. All USART interfaces can be served by the DMA controller.

1.1.1.7. Serial peripheral interface (SPI)

Up to two SPIs can communicate up to 18 Mbits/s in slave and master modes in fullduplex and simplex communication modes. The 3-bit prescaler gives 8 master mode frequencies and the frame is configurable to 8 bits or 16 bits. The hardware CRC generation/verification supports basic SD Card/MMC modes. Both SPIs can be served by the DMA controller.

1.1.1.8. GPIOs (general-purpose inputs/outputs)

Each of the GPIO pins can be configured by software as output (push-pull or open-drain), as input (with or without pull-up or pull-down) or as peripheral alternate function. Most of the GPIO pins are shared with digital or analog alternate functions. All GPIOs are high current capable. The I/Os alternate function configuration can be locked if needed following a specific sequence to avoid spurious writing to the I/Os registers. I/Os on APB2 with up to 18 MHz toggling speed.

1.1.1.9. ADC (analog-to-digital converter)

Two 12-bit analog-to-digital converters are embedded into STM32F103xx performance line devices and each ADC shares up to 16 external channels, performing conversions in single shot or scan modes. In scan mode, automatic conversion is performed on a selected group of analog inputs. Additional logic functions embedded in the ADC interface allow:

- Simultaneous sample and hold
- Interleaved sample and hold
- Single shunt

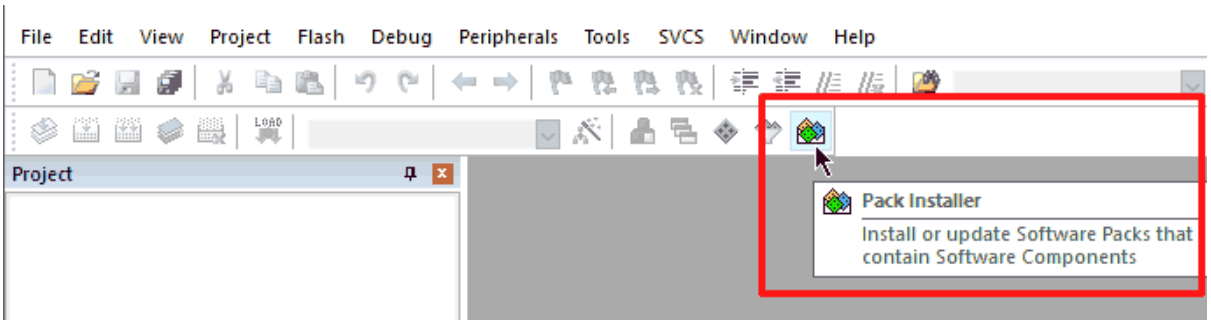
1.2. Keil - IDE Setup

1.2.1. Installing prerequisite STM32 Keil software pack

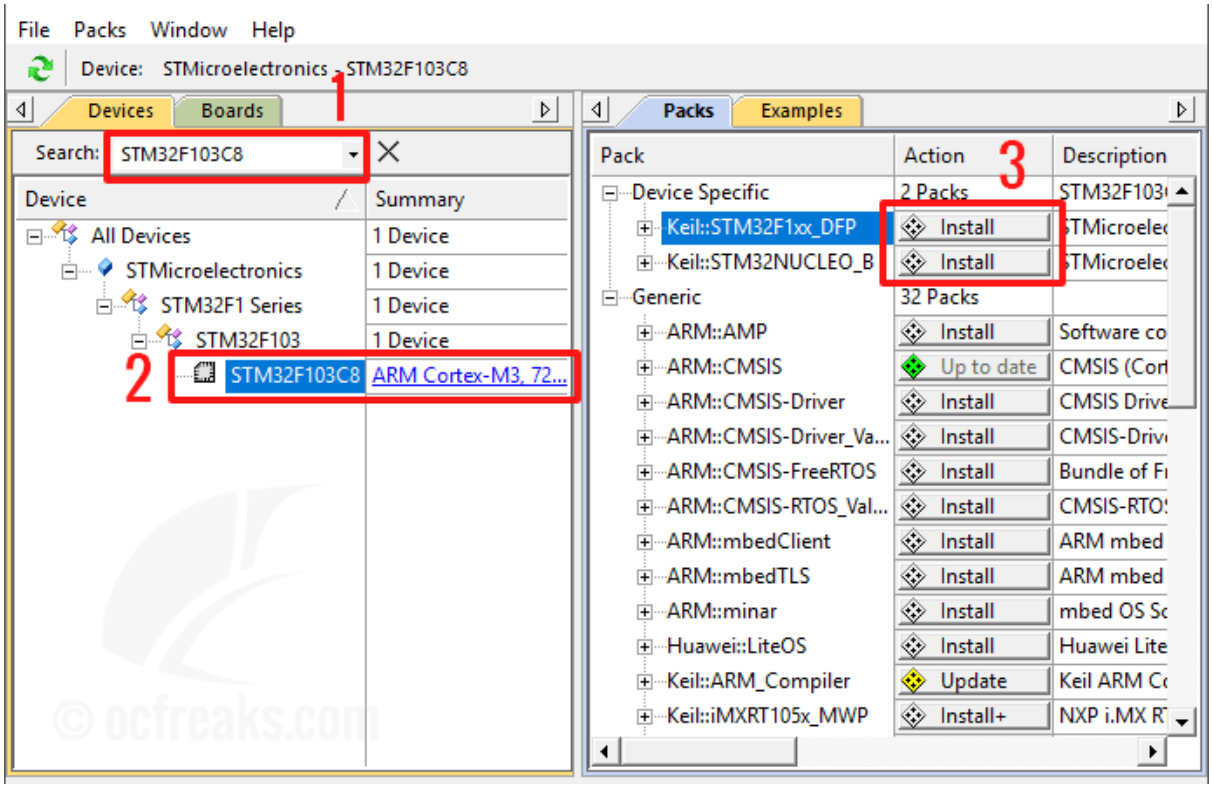
Step A. Download latest Keil MDK uVision5 from Keil's website.

Step B. Install Keil uVision 5 to default path.

STEP C. Open Keil 5 and click on "Pack Installer" icon as shown below:



STEP D. On the left half on the window, under "Devices" type "STM3F103C8" (or other device name depending on the device present on your development board) in search box and select the MCU in the list below. Now, on the right half of the window click on the "install" button which is towards to the right of "Keil:STM32F1xxx_DFP" and "Keil:STM32NUCLEO_B". Repeat this step if want to add support for other device family. After this, wait until pack installer finishes downloading the required pack files for selected MCU.



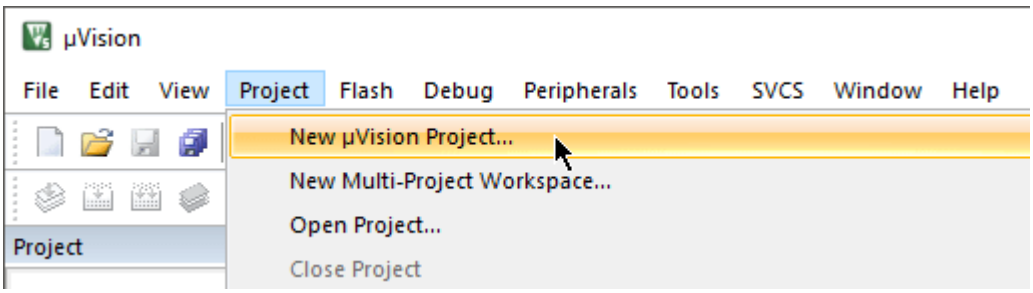
Alternatively, you can manually download the software pack and install it directly from MDK5 Software Packs. It will be present Under “KEIL-> STMicroelectronics STM32F1 Series Device Support, Drivers”. In general, for STM32Fx Devices.

STEP E. After installing from Pack Installer, you will get a confirmation to reload packs. Click “Yes”.

1.2.2. Create a new project.

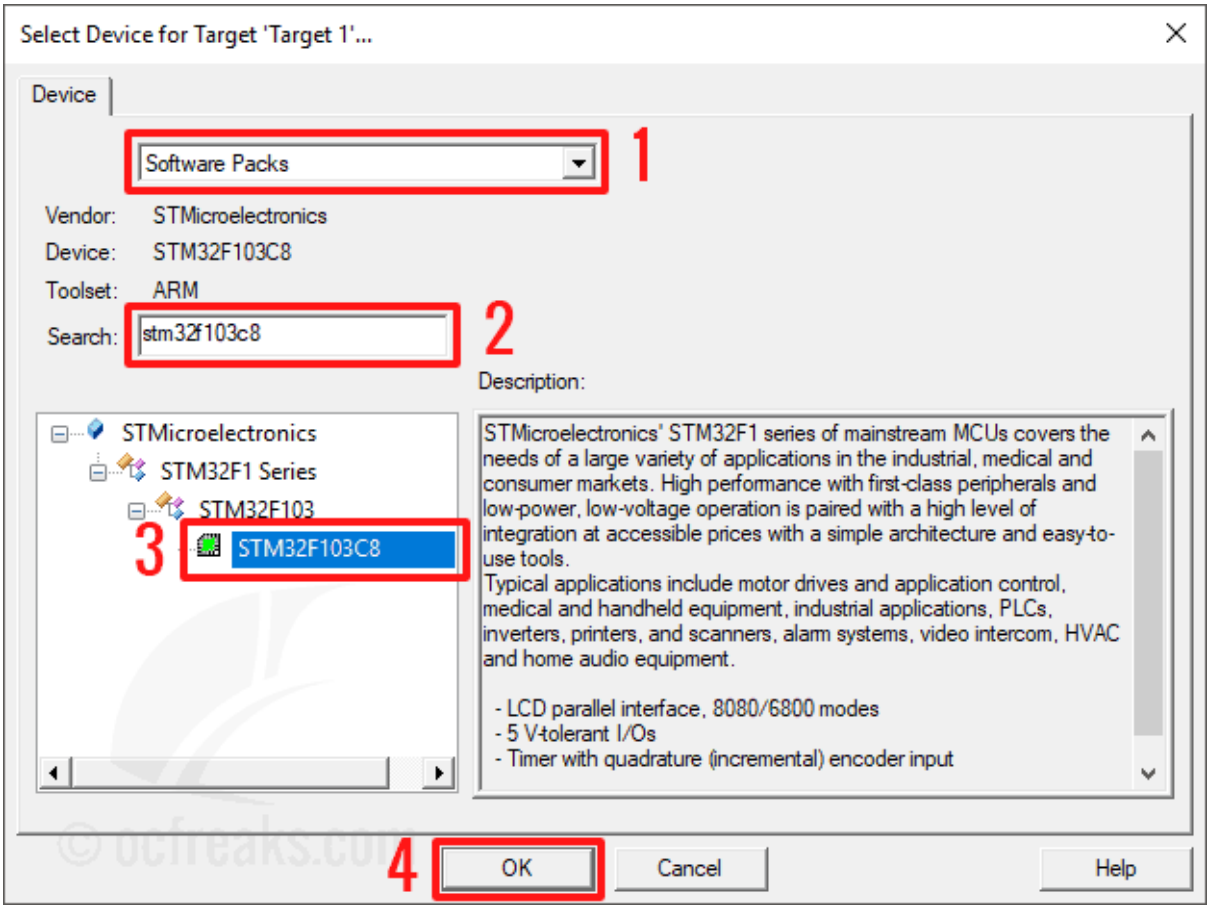
Okay, so now we have the necessary packs installed to create our first STM32 project in Keil 5. Just follow the steps mentioned below to create a new project in Keil uV 5 or if your project is not working properly:

Step 1. Open the Keil IDE, under main menu go to “Project->New uVision Project…” and a window prompt will open asking to save the new project. Type your desired project name and save.



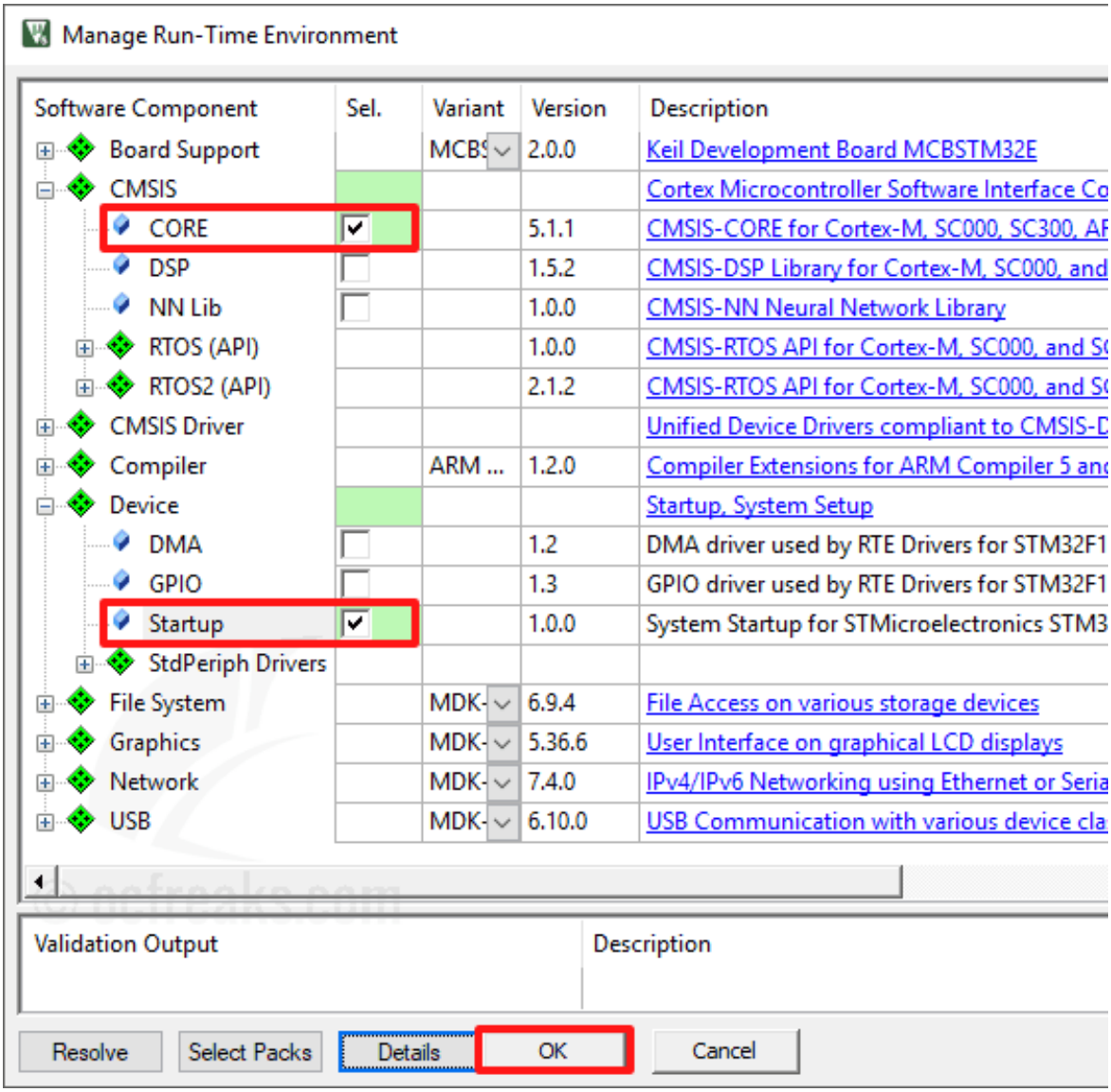
Step 2. After that, a new window will appear as shown below. Make sure "Software Packs" is selected for the 1st drop down. In the search box below it, type "STM32F103C8" and then select the device from list below. For e.g.: STM32F103C8 for STM32 Blue Pill, STM32F103RB for Nucleo-F103RB, STM32F030R8 for Nucleo-F030R8 and so on.

Finally click "OK".

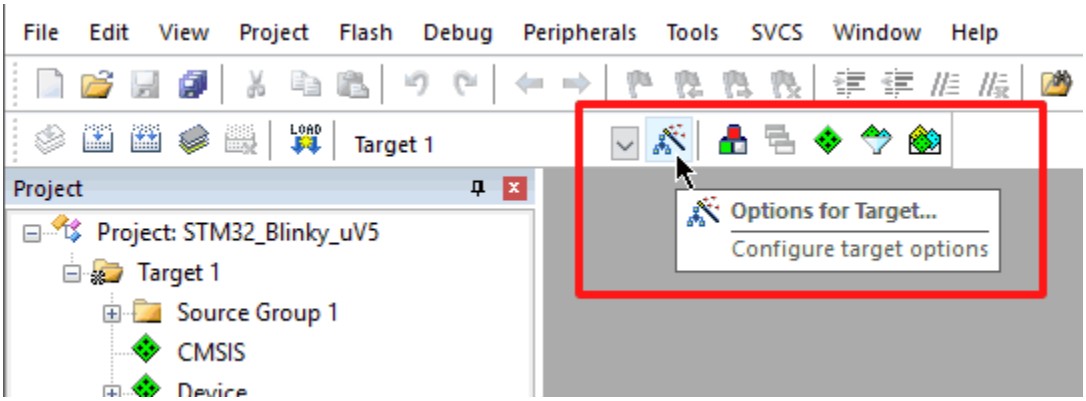


Step 3. A. For CMSIS: Inside the "Manage Run-Time Environment Window" select the check boxes for "CORE"

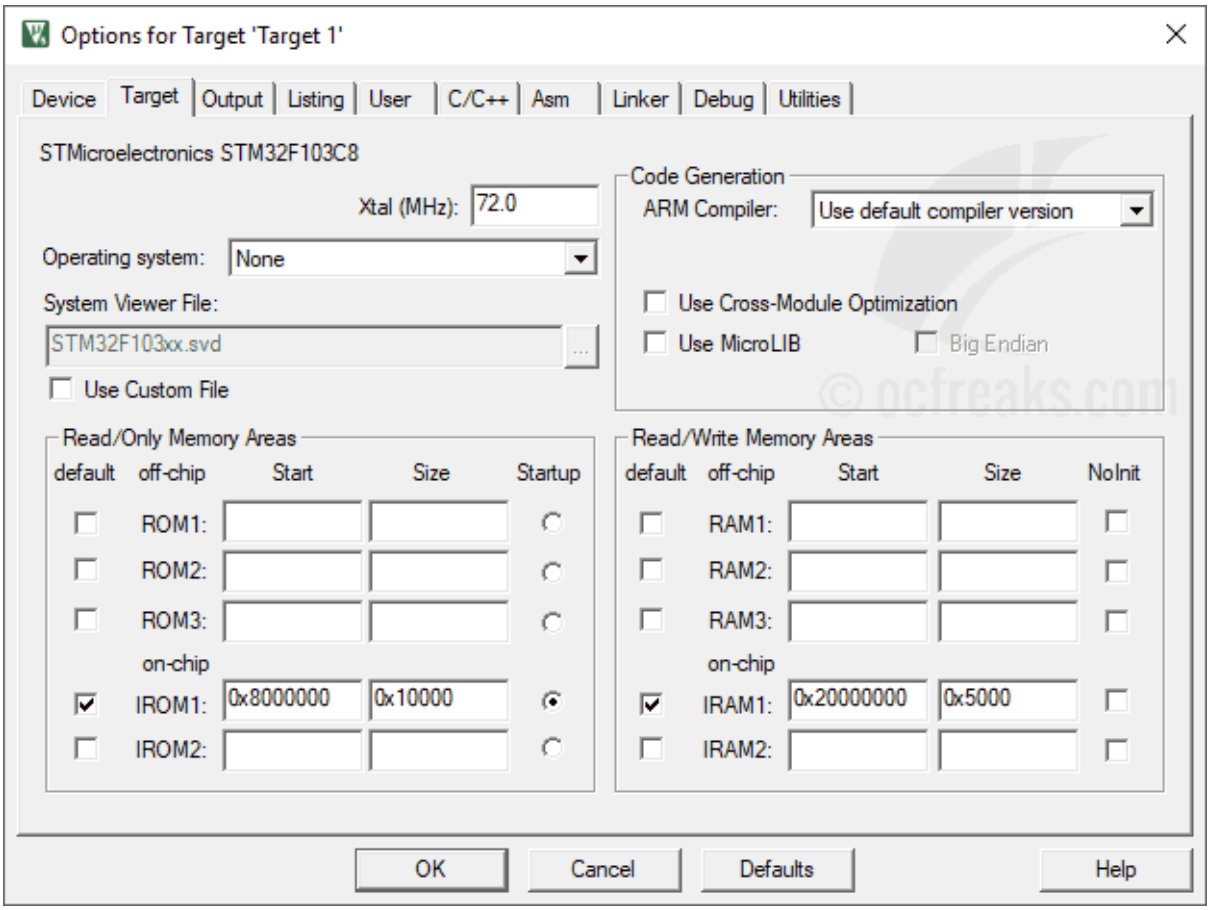
under “CMSIS” and “Startup” under “Device”. If you want to select any other libraries, you can do so by selecting the respective checkboxes. Selecting “Startup” will automatically add all the necessary startup/boot files required for STM32F1xx device, so we don’t have to import them from external sources. The selection of libraries can be changed any time later.



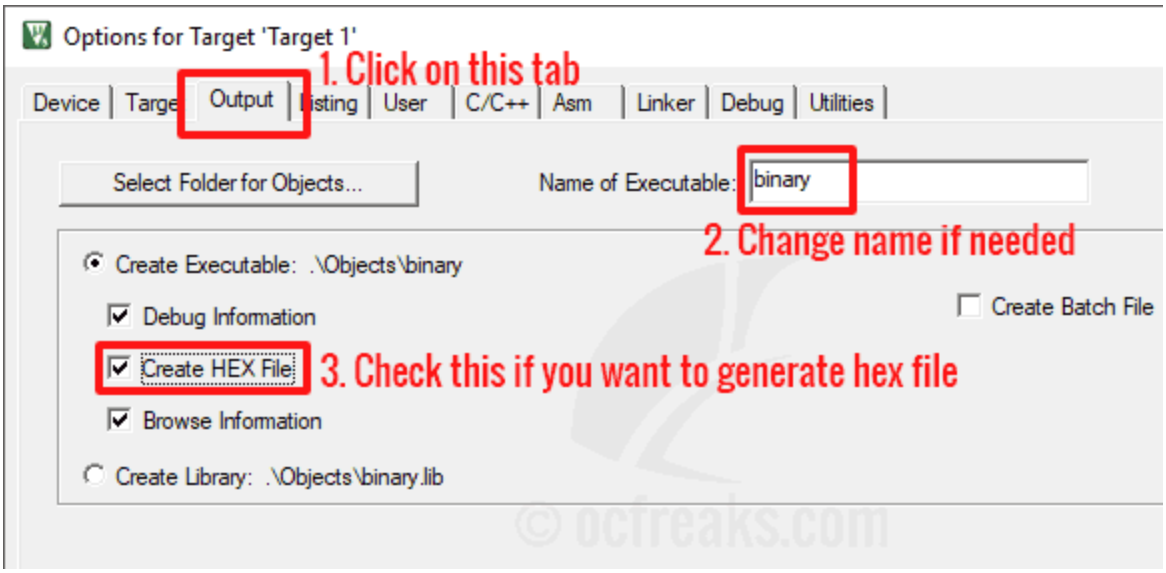
Step 4. Now click on “Options for Target” button as shown below:



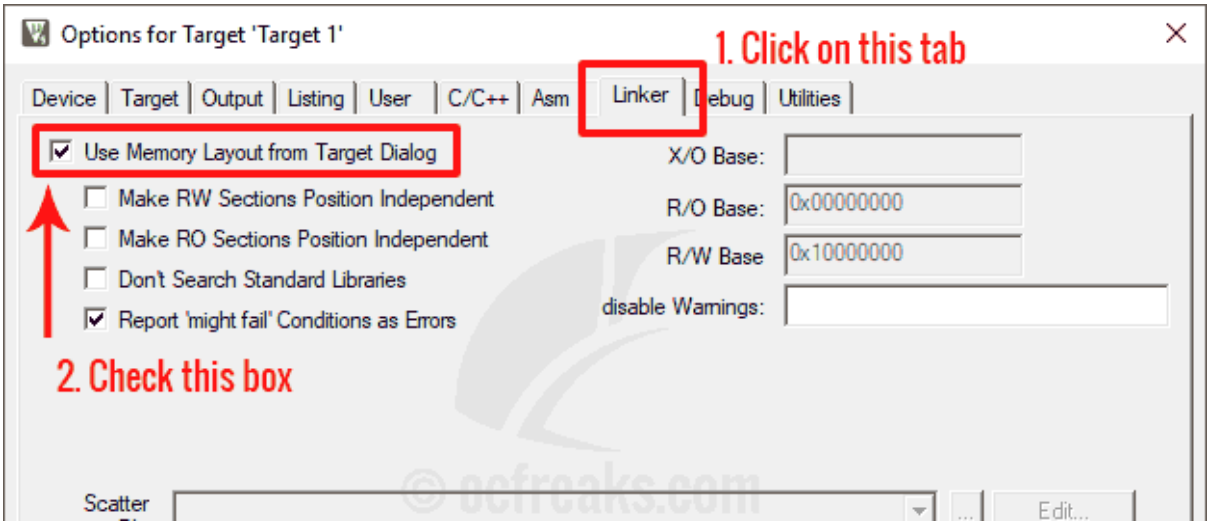
Make sure the settings match as shown below.



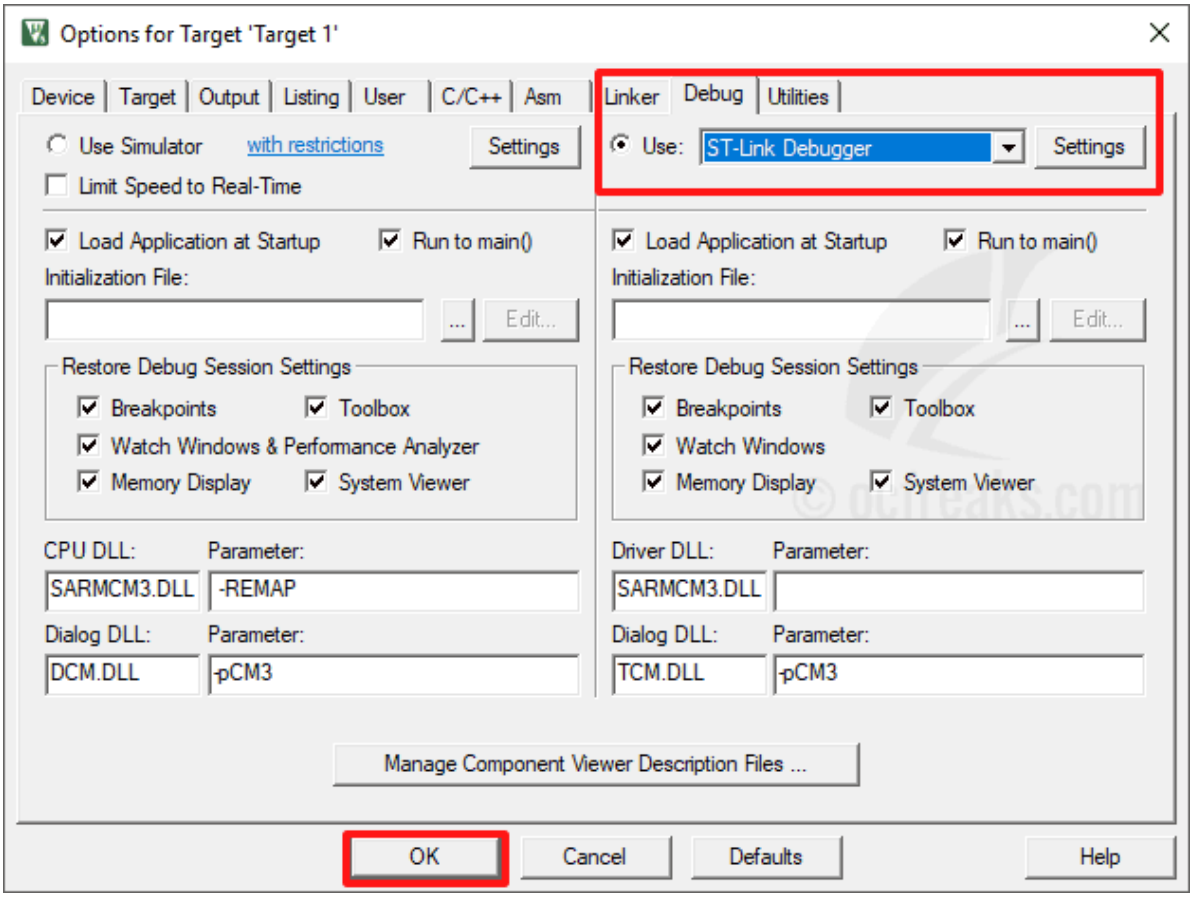
Step 5. Now, click on the “Output” tab. If you want to generate hex file then you can check “Create HEX File”. You also enter a suitable name for the executable output file.



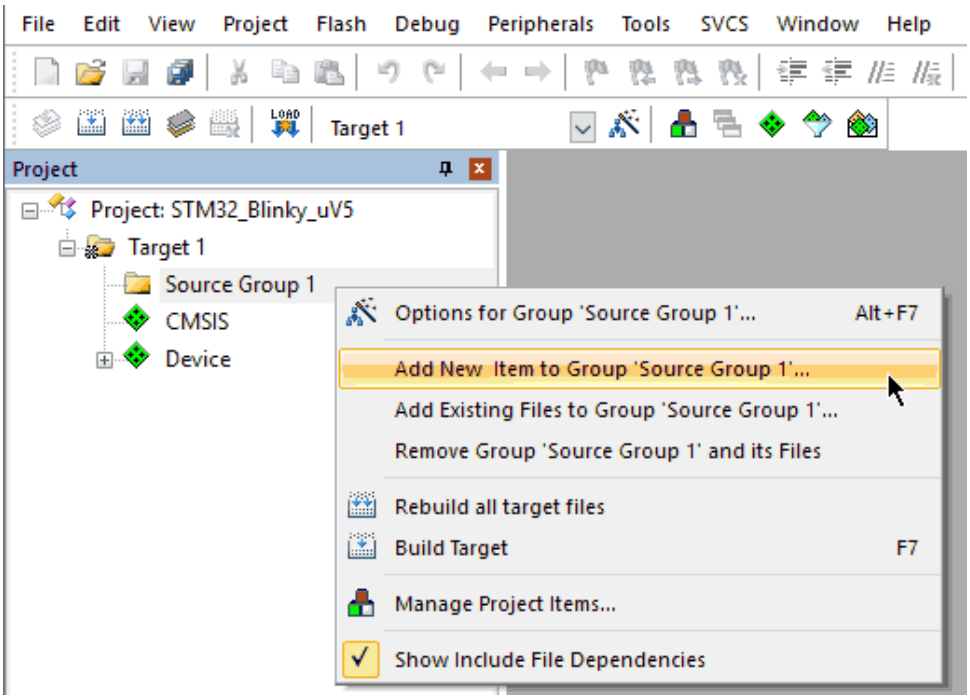
Step 6. Then click on the “Linker” tab and under that tab check the checkbox option which says, “Use Memory Layout from Target Dialog”.



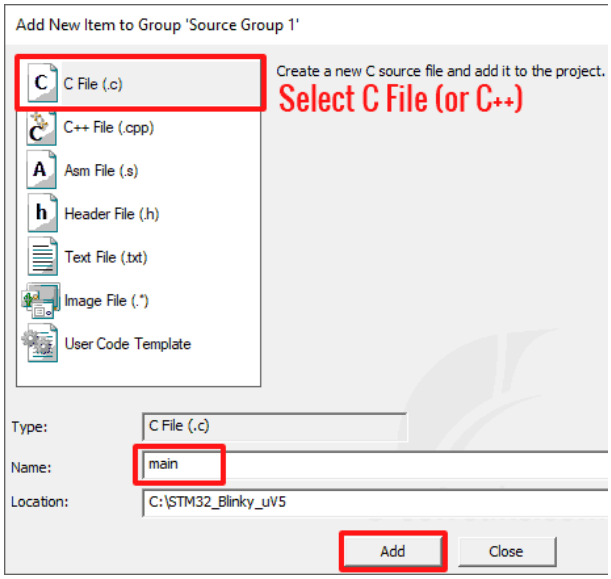
Step 7. Now, under the “Debug” tab, select ST-LINK as debugger since it’s the most common for debugging and programming STM32. Finally, to click “OK” to apply settings and close window.



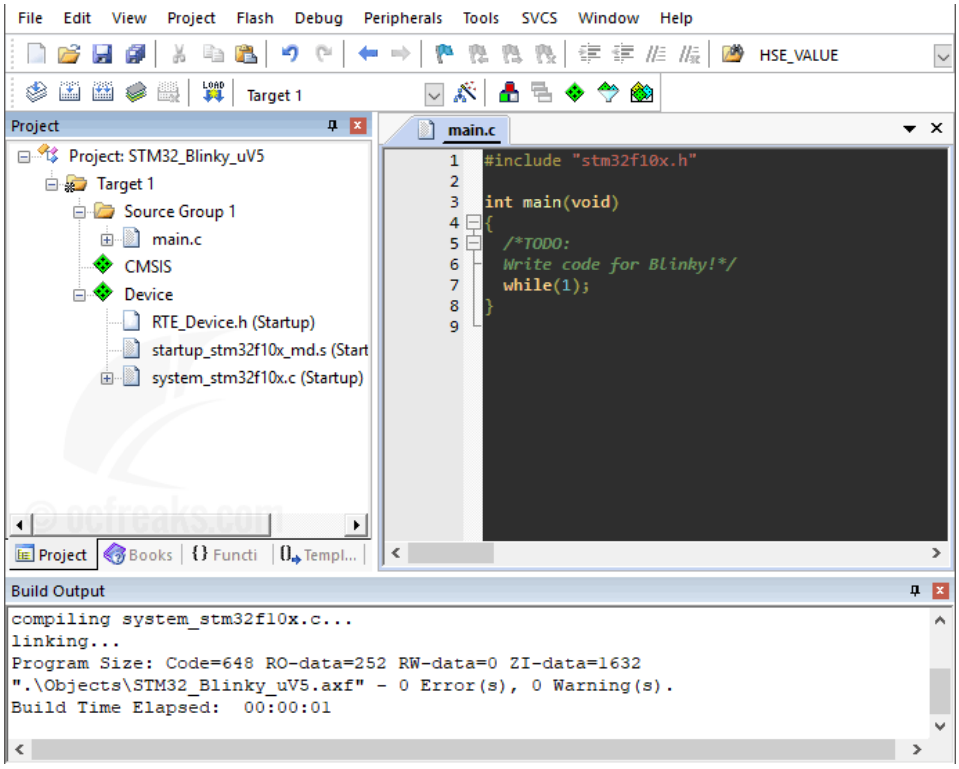
Step 8. Now, in the source navigation pane on the left area, right click on “Source Group 1” and select “Add New Item to Group ‘Source Group 1’”.



Step 9. A new window will pop-up to add an item as shown below. Select “C File (.c)” or C++ File (.cpp), then enter the name of the file in the text box to the right of “Name:” and click “Add”.



Step 10. Now you can write your code in the editor. To compile your program Press “F7” key or in the main menu go to “Project->Build Target”. To check for any compilation errors, you can have a look at the build output at the bottom of main window.



1.3. Bootloader

1.3.1. Microcontroller Programming

the programming or flashing process is the process of updating the content of the flash memory and since the flash is used to store the program code the process is also called programming or burning. Depending on the position of the flash driver different programming techniques are found:

- Off- Circuit Programming.
- In-Circuit Programming.
- In-Application Programming.

1.3.1.1. The In-Circuit Programming

method is used to update the entire contents of the Flash memory, using the JTAG, SWD protocol or the boot loader to load the user application into the microcontroller. ICP offers quick and efficient design iterations and eliminates unnecessary package handling or socketing of devices.

1.3.1.2. In-Application Programming

In contrast to the ICP method, in-application programming (IAP) can use any communication interface supported by the microcontroller (I/Os, USB, CAN, UART, I2C, SPI, etc.) to download programming data into memory. IAP allows the user to re-program the Flash memory while the application is running. Nevertheless, part of the application has to have been previously programmed in the Flash memory using ICP. Steps:

- Receive the file.
- Flash the file.

1.3.2. Bootloader Overview

The bootloader is the first code that is executed after a system reset. Its goal is to bring the system to a state in which it can perform its main function. This requires hardware initialization and choosing the correct image to load from flash. Because of its key role, the bootloader is usually placed in a part of the Flash that is protected from accidental erasure or corruption.

1.3.2.1. Why is bootloader needed?

It is beneficial to be able to update a deployed system (to fix bugs or just to update with new feature). Access to the pcb its self is in many cases not possible and the connector might not even be on it since the pin are used for something else .

1.3.2.2. Where is bootloader stored?

Bootloader reside in the Rom or flash memory of the Mcu usually write protected so you can't destroy them.

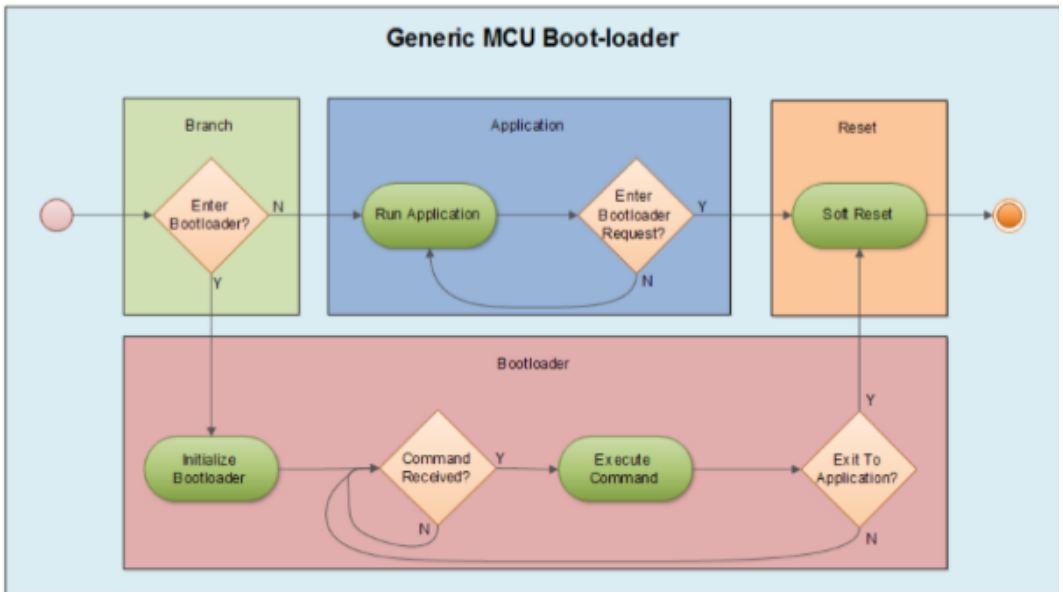
1.3.2.3. Bootloader Requirements

Each bootloader will have its own unique set of requirements based on the type of application; however, there are still a few general requirements that are common to all bootloaders. They are:

1. Ability to switch or select the operating mode (Application or bootloader).
2. Communication interface requirements (USB, CAN, I2C, USART, etc.).
3. Flash system requirements (erase, write, read, location).
4. Application checksum (verifying the app is not corrupt).
5. Code Security (Protecting the bootloader and the application).

1.3.2.4. The Bootloader System

Bootloaders can come in many different sizes and in many different flavors but in general the operation of a system with a bootloader is relatively standard. There are three major components to these systems that can be seen in the Figure. They are the branching code (green), the application code (blue) and the bootloader code (red). For most systems we prefer them to be executing the application path most of their



operational life. Figure (1) highlights the execution path to get to the application by the dashed red line. The orange block is a common block used by both the bootloader and the application to reset the system.

The branching code (green) makes the decision as to whether the bootloader is loaded or whether the application is loaded. The branching code in a very simple system can be nothing more than checking a GPIO for a certain state to enter the bootloader; however, such basic systems are usually only used by chip manufacturers in order to demonstrate the functionality of the chip and is usually not recommended if a robust boot-loading solution is required. In a more complex system, the bootloader may completely load itself into memory and perform some basic system functions for a short period of time before making the branching decision possibly to perform a system integrity check. For this reason, in most cases the branching code is included in the bootloader. The application code (blue) is executed only after the branching code has made the decision that there are no requests to enter the bootloader and that the application code is safe to execute.

Even while the application is loading and performing its duties, it should be designed so that it can accept a command to enter the bootloader. When the application receives such a request the application will perform any cleanup that needs to be performed and when it is safe to do so the application will perform a soft reset of the system. In most cases this is done by writing incorrect values to the watchdog timer that in turn will then cause a power-on-reset of the system. Additional details of how to make the branching code recognize that there has been a request to enter the bootloader will be discussed later in the paper. The bootloader code (red) performs all the functions of the bootloader. When the bootloader is loaded into memory it will begin initializing a basic subset of peripherals that are required for the bootloader to perform all its functions. These peripherals are typically the system clock, interrupt service routines and tables, a communication peripheral and perhaps a state-machine or basic task scheduler. This allows the bootloader to communicate with the outside world and accept commands to perform flashing instructions.

1.3.3. Bootloader Types

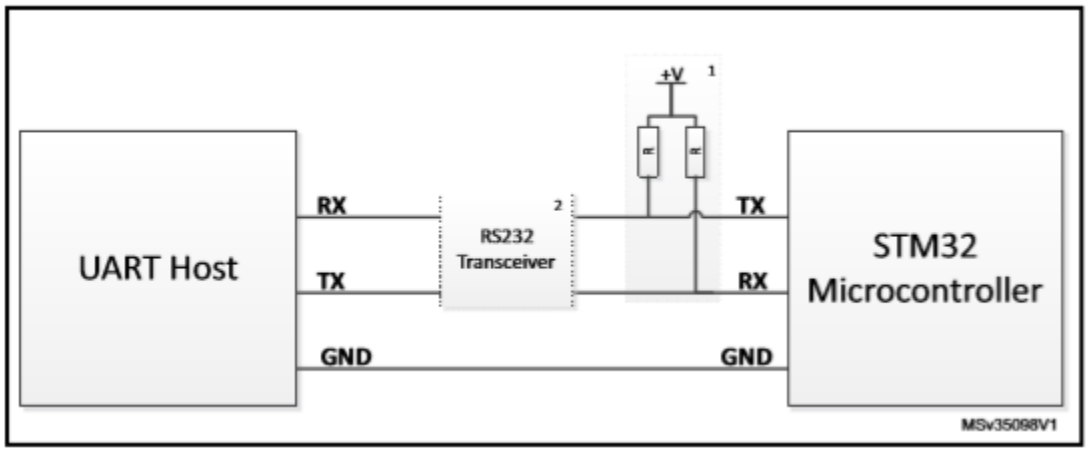
Bootloader is specially designed to understand communications from the outside world via any number of communications protocols like:

- Uart
- I2C
- SPI
- CAN
- LIN
- Eherent

Based on these protocols we have different types of bootloaders uses different types of communications protocols.
A bootloader (sometimes called a boot manger) has unique architecture of the embedded processor it run on and includes some application specific Aspects.

1.3.4. UART Bootloader

It's a bootloader which receives the image via Uart communication protocol like stm32 bootloader. To use the USART bootloader, the host must be connected to the RX and TX pins of the desired USARTx interface via a serial cable.



1.3.5. OTA Update

An over-the-air (OTA) update is the wireless delivery of new software.
OTA updates are a more efficient way for OEMs to fix bugs and update software than to manually upgrade each individual device. OTA updates can catch issues before devices

launch, which can save OEMs time and money, as well as reduce the software development and quality-assurance (QA) process.
OTA updates also enable OEMs to update software more easily on devices that are difficult to access, such as advertising displays.
There are some security risks involved with OTA updates. OEMs that do not properly secure their OTA update process can enable hackers to change the device's software. Attackers can push unofficial updates that introduce security vulnerabilities. For example, an OTA company called Adups, based in China, installed adware and spyware to collect personal information from devices.

1.3.5.1. OTA Requirements

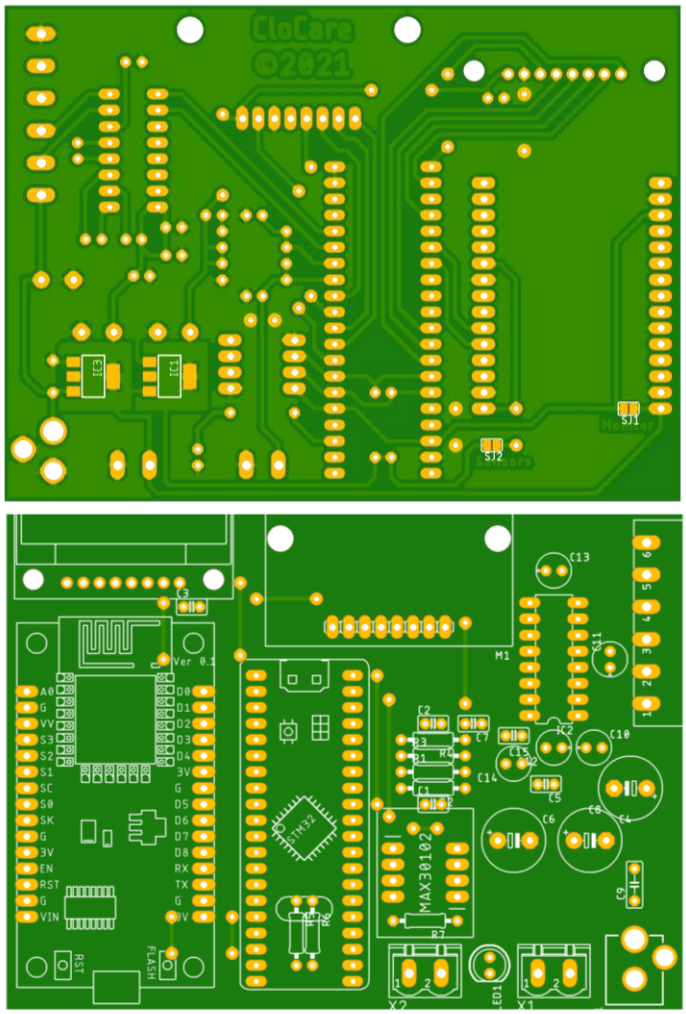
- Safety
  - no image can be downloaded until it has been verified as the correct image for the specific ECU subcomponent; utilizing CRCs, checksums, and other release version compatibilities used.
- Security
  - data should be sent utilizing security protocol and encryption mechanisms.
- Flexibility
  - Should support national and regional requirements.
  - Should provide an efficient mechanism to support multiple car models and car groups.

1.3.5.2. OTA Update Benefits

- Cost efficient. OEMs can seamlessly manage firmware updates across a fleet of IoT devices from one unified interface. The costs significantly decrease over the entire lifecycle of a car.
- Continuous improvements. Bugs can be fixed, and product behavior can be enhanced after the device lands in the hands of your consumers. This can potentially eliminate costly recalls and in-person maintenance.
- Improved scalability. FOTA updates enable manufacturers to add new features to infrastructure after the release without physical access to upgrade firmware.
- Faster time-to-market. Developers can test new features on selected devices and deploy frequently knowing that the products will remain stable. Firmware updates can be dispatched even while the vehicle is still on the production line or in dealer center.

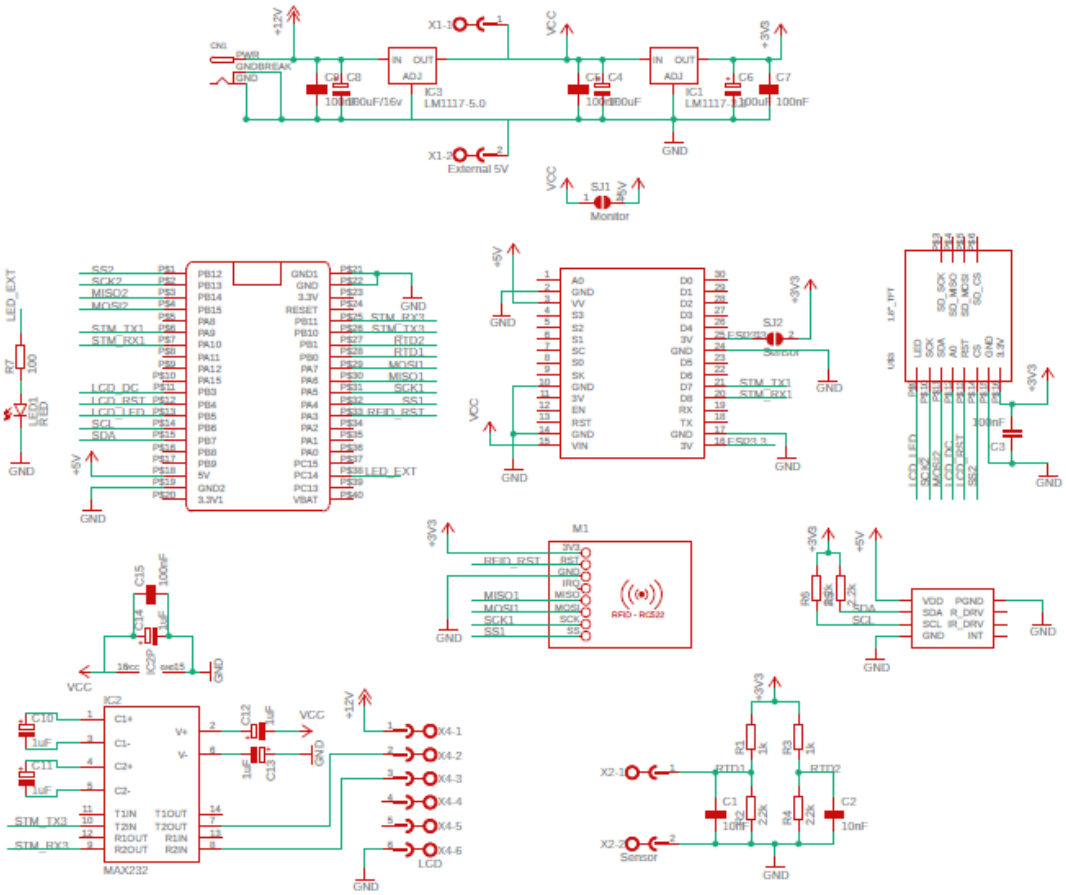
- Improved safety and compliance. OEMs can use SOTA updates to patch the known vulnerabilities, e.g., defective adapter plugs, instead of recalling the vehicles. Your business can respond to the legal and regulatory responsibilities faster and more cost-effectively.
- Better software quality. SOTA updates allow you to continuously amend your systems. Whenever you discover a new opportunity for improvement (e.g., a way to reduce vehicles fuel consumption), you can instantly deliver it to customers, instead of waiting to incorporate it in a new batch of vehicles.
- Timely updates. Ensure that your users are receiving the latest products from your business — new add-ons to the infotainment systems or navigation. For autonomous driving, regular OTA software updates will become crucial to ensure safe and smooth navigation and routing.
- Two-way communication. OTA data exchanges can happen both ways. By collecting data on the vehicle usage or performance and deploying analytics tools, OEMs can promote better customer experience and show customers that they care about them by issuing regular system updates. Additionally, you can gather intel for R&D and ensure preventive maintenance. Finally, you can also monetize your in-vehicle generated data using Caruso Dataplace — an innovative data marketplace for the mobility ecosystem. And you can enrich your solutions with additional data sources or white-label solutions purchased through this service.

1.4. Schematic & PCB design



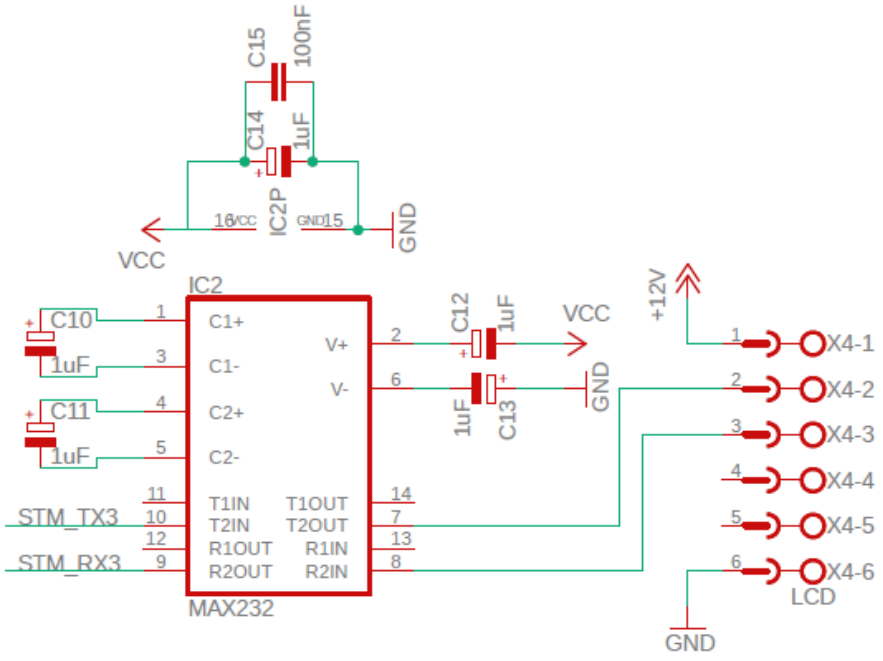
In this section, we will explore the PCB design and hardware components used to implement the embedded devices in our solution from the schematic perspective.

The figure below is the full schematic of the PCB board used for both devices. Let's break it down to pieces to see what each part is about and how the same board is used for both devices.



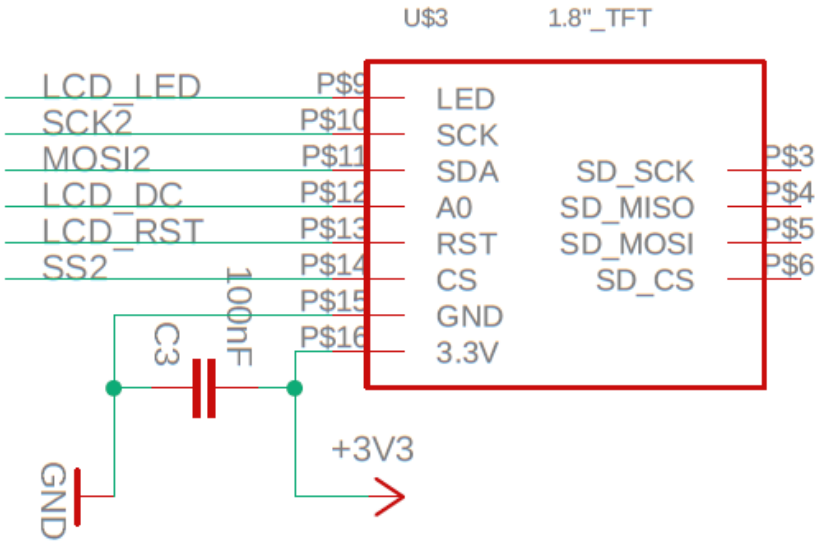
1.4.1. Touch Screen LCD

The 7 inches touch screen used in the registration device is powered using 12v DC power source and can communicate with the microcontroller via UART module. As for the microcontroller can only tolerate 5v, a MAX232 IC is used to act as a bridge between the microcontroller and the LCD.



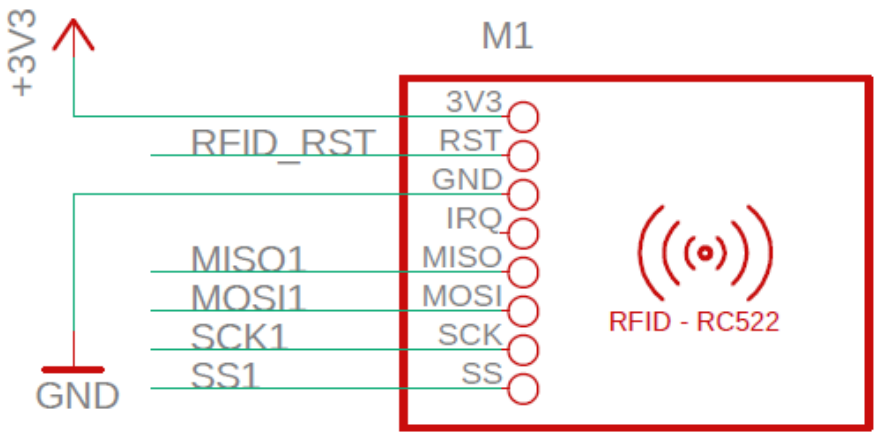
1.4.2. TFT Screen

The TFT used in the monitoring device is powered using 3.3v DC source and communicates with the STM32F103C8 microcontroller using SPI protocol. Besides the regular SPI wires, it also contains a pin for LED turning on and off, a RST button, and a Data/Command line.



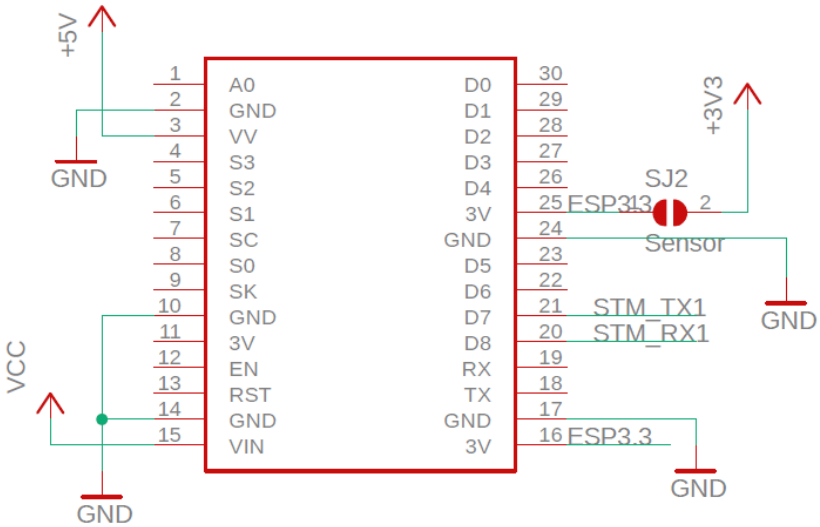
1.4.3. RFID

The RFID reader is essential component for both devices as it's used to register, recognize, or login a patient to the system. It's connected to the microcontroller for power supply and uses an SPI protocol for data communication. It uses a different SPI channel from the one used for the TFT. Besides the normal SPI lines, it has a reset line used for various commands.



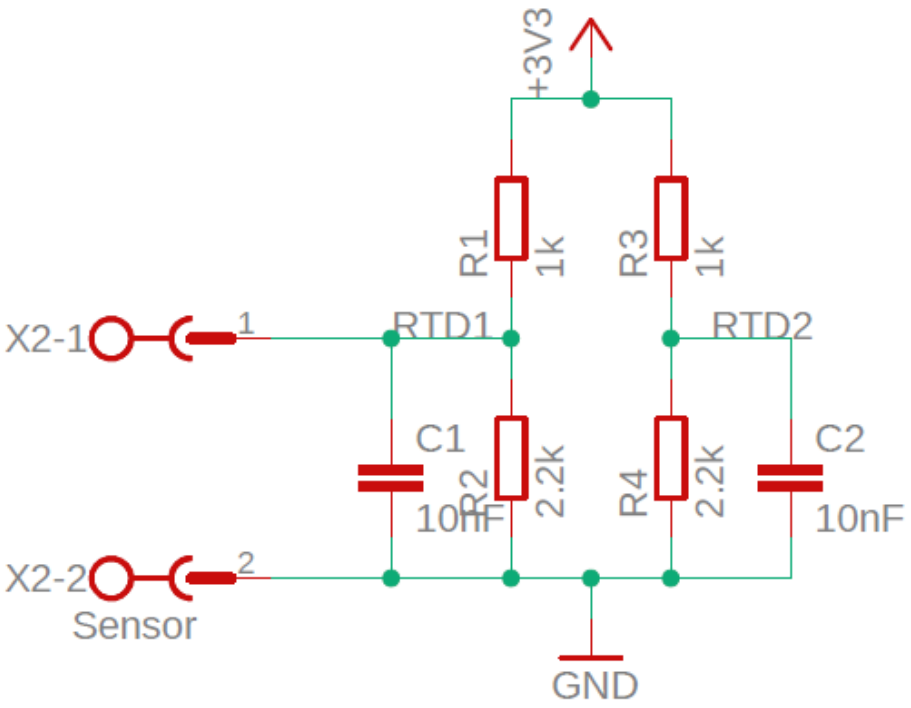
1.4.4. NodeMCU WiFi module – ESP8266

The NodeMcu is a key component in the system, it's used to communicate data between the STM32F103C8 microcontroller and the local server. It also provides a 5v supply line from the outside word to the other components in the circuit. It uses UART to communicate with the microcontroller.



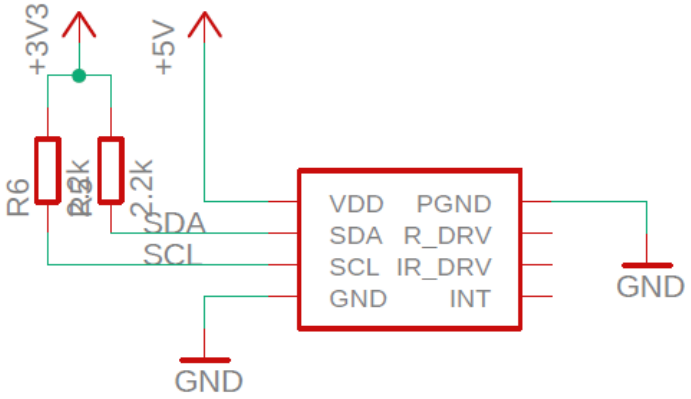
1.4.5. Temperature Sensor

The Thermocouple theorem is basically based on a voltage divider, the reading from the sensor is converted twice and an average reading is calculated. So the sensor uses 2 ADC pins along with 3.3v DC power source.



1.4.6. SPO2 Sensor

The MAX30102 module is basically an i2c slave. It can work with or without interrupts. Here we use it without interrupts. The module is powered by 3.3v but it can tolerate 5v as well. The i2c lines are pulled up using 3.3v power line.



1.4.7. Conclusion

The PCB board contains wiring for each peripheral used in both devices without interleaving connection. Since the basics are the same, we can choose to wire the LCD touch screen or the TFT with the sensors to provide either the registration device capability or the monitoring.

1.5. Microcontroller Drivers

Over this chapter we'll take a quick overview on the microcontroller drivers implemented to make the project functional.

1.5.1. RCC

1.5.1.1. RCC Introduction

Reset and clock control(RCC) peripheral is used to control the clock source of the internal peripheral as well as reset signal. They are used as clock source for hardware blocks either directly or indirectly via pll that allow to achieve high frequencies.

1.5.1.2. RCC Main Features

- Enable clock on the peripheral to work.
- The Rcc gets several internal (LSI, HSI and CSI)and external(LSE and HSE) clocks.
- Conrtol reset signal.

1.5.2. GPIO

1.5.2.1. GPIO Introduction

The general purpose I/O ports (GPIO) can be used for driving load , reading digital and analog signal ,controlling external components , generating triggers for external devices etc. Each of the general-purpose I/O ports has two 32-bit configuration registers (GPIOx_CRL, GPIOx_CRH), two 32-bit data registers (GPIOx_IDR, GPIOx_ODR), a 32-bit set/reset register (GPIOx_BSRR), a 16-bit reset register (GPIOx_BRR) and a 32-bit locking register (GPIOx_LCKR).

1.5.2.2. GPIO Main Features

- each port bit of the General Purpose IO (GPIO) Ports, can be individually configured by software in several modes: (Input floating - Input pull-up -

Input-pull-down - Analog - Output open-drain - Output push-pull -Alternate function push-pull - Alternate function open-drain)

- Each I/O port bit is freely programmable; however the I/O port registers have to be accessed as 32-bit words (half-word or byte accesses are not allowed).
- The purpose of the GPIOx_BSRR and GPIOx_BRR registers is to allow atomic read/modify accesses to any of the GPIO registers. This way, there is no risk that an IRQ occurs between the read and the modify access.
- All GPIO pins have an internal weak pull-up and weak pull-down that can be activated or not when configured as input.

1.5.3. ADC

1.5.3.1. ADC Introduction

The 12-bit ADC is a successive approximation analog-to-digital converter. It has up to 18 multiplexed channels allowing it measure signals from sixteen external and two internal sources. A/D conversion of the various channels can be performed in single, continuous, scan or discontinuous mode. The result of the ADC is stored in a left-aligned or right-aligned 16-bit data register.The analog watchdog feature allows the application to detect if the input voltage goes outside the user-defined high or low thresholds.

1.5.3.2. ADC Main Features

- 12-bit resolution
- Interrupt generation at End of Conversion, End of Injected conversion, and Analog watchdog event
- Single and continuous conversion modes
- Scan mode for automatic conversion of channel 0 to channel ‘n’
- Self-calibration
- Data alignment with in-built data coherency
- Channel by channel programmable sampling time
- External trigger option for both regular and injected conversion
- Discontinuous mode
- Dual mode (on devices with 2 ADCs or more)

1.5.4. IWDG

1.5.4.1. IWDG Introduction

The devices have two embedded watchdog peripherals which offer a combination of high safety level, timing accuracy and flexibility of use. Both watchdog peripherals (Independent and Window) serve to detect and resolve malfunctions due to software failure, and to trigger system reset or an interrupt (window watchdog only) when the counter reaches a given timeout value. The independent watchdog (IWDG) is clocked by its own dedicated low-speed clock (LSI) and thus stays active even if the main clock fails. The window watchdog (WWDG) clock is prescaled from the APB1 clock and has a configurable time-window that can be programmed to detect abnormally late or early application behavior.

1.5.4.2. IWDG Main Features

- Free running down counter.
- clocked from an independent RC oscillator (can operate in Standby and Stop modes)
- Reset (if watchdog activated) when the down counter value of 0x000 is reached.

1.5.5. USART

1.5.5.1. USART Introduction

USART is a universal serial communication bus that can implement full-duplex asynchronous serial communication. In embedded system design, it is usually used for communication between the control system and peripheral devices, including controller and terminal equipment, etc.

1.5.5.2. USRAT Main Features

- It can work in both synchronous & asynchronous modes.
- Full-duplex operation (receive data and send data independently of each other).
- Independent high-precision baud rate generator does not occupy the timer / counter.
- Supports 5, 6, 7, 8 and 9 data bits, 1 or 2 stop bit serial data frame structure.
- Parity bit generation and verification supported by hardware.
- Data overflow detection Frame error detection.
- Three completely independent interrupts, TX transmission completed, TX transmission data register empty, RX reception completed.

1.5.5.3. USART driver code

```
void UART1_voidInit()
{
  // Baud Rate selection 115200
  USART1_BRR = 0x46;
  // Clear configuration bits
  USART1_CR1 = 0;
  // Enable TX - RX
  SET_BIT(USART1_CR1, USART_CR1_TE);
  SET_BIT(USART1_CR1, USART_CR1_RE);
  // Enable UART
  SET_BIT(USART1_CR1, USART_CR1_UE);
  // Clear Status Register
  USART1_SR = 0;
}

void UART1_voidDeInit()
{
  // Clear Baud Rate selection
  USART1_BRR = 0;
  // Clear configuration bits
  USART1_CR1 = 0;
  // Clear Status Register
  USART1_SR = 0;
}
void UART1_voidTransmitSync(u8 cpy_arr[], u8 cpy_arrLen)
{
  u16 i = 0;
  while (i != cpy_arrLen)
  {
    USART1_DR = cpy_arr[i];
    while (GET_BIT(USART1_SR, USART_SR_TC) == 0)
      ;
    CLR_BIT(USART1_SR, USART_SR_TC);
    i++;
  }
}

void UART1_voidRecieveSync(u16 cpy_data_length, u8 *DataRecieved)
{
  u16 local_idx = 0;
  while (local_idx < cpy_data_length)
  {
    while (GET_BIT(USART1_SR, USART_SR_RXNE) == 0)
      ;
```

```
    CLR_BIT(USART1_SR, USART_SR_RXNE);
    DataRecieved[local_idx++] = 0xFF & USART1_DR;
  }
}
```

1.5.6. SPI

1.5.6.1. SPI Introduction

The serial peripheral interface (SPI) allows half/ full-duplex, synchronous, serial communication with external devices. The interface can be configured as the master and in this case, it provides the communication clock (SCK) to the external slave device. The interface is also capable of operating in multi-master configuration. It may be used for a variety of purposes, including simplex synchronous transfers on two lines with a possible bidirectional data line or reliable communication using CRC checking.

1.5.6.2. SPI Main Features

- Full-duplex synchronous transfers on three lines
- Simplex synchronous transfers on two lines with or without a bidirectional data line
- 8- or 16-bit transfer frame format selection
- Master or slave operation
- Multimaster mode capability
- 8 master mode baud rate prescalers (fPCLK/2 max.)
- Slave mode frequency (fPCLK/2 max)
- Faster communication for both master and slave
- NSS management by hardware or software for both master and slave: dynamic change of master/slave operations
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Dedicated transmission and reception flags with interrupt capability
- SPI bus busy status flag
- Hardware CRC feature for reliable communication:
    - CRC value can be transmitted as last byte in Tx mode.
    - Automatic CRC error checking for last received byte
- Master mode fault, overrun and CRC error flags with interrupt capability.
- 1-byte transmission and reception buffer with DMA capability: Tx and Rx requests

1.5.7. I2C

1.5.7.1. I2C Introduction

I2C (inter-integrated circuit) bus Interface serves as an interface between the microcontroller and the serial I2C bus. It provides multimaster capability, and controls all I2C bus-specific sequencing, protocol, arbitration, and timing. It supports the standard mode (Sm, up to 100 kHz) and Fm mode (Fm, up to 400 kHz). It may be used for a variety of purposes, including CRC generation and verification, SMBus (system management bus) and PMBus (power management bus). Depending on specific device implementation DMA capability can be available for reduced CPU overload.

1.5.7.2. I2C Main Features

- Parallel bus / I2C protocol converter
- Multimaster capability: the same interface can act as Master or Slave.
- I2C Master features:
    - Clock generation.
    - Start and Stop generation.
- I2C Slave features:
    - Programmable I2C Address detection.
    - Dual Addressing Capability to acknowledge 2 slave addresses.
    - Stop bit detection.
- Generation and detection of 7-bit/10-bit addressing and General Call.
- Supports different communication speeds:
    - Standard Speed (up to 100 kHz).
    - Fast Speed (up to 400 kHz).
- Analog noise filter.
- Status flags:
    - Transmitter/Receiver mode flag.
    - End-of-Byte transmission flag.
    - I2C busy flag.
- Error flags:
    - Arbitration lost condition for master mode.
    - Acknowledgment failure after address/ data transmission.
    - Detection of misplaced start or stop condition.
    - Overrun/Underrun if clock stretching is disabled.
- 2 Interrupt vectors:

  - 1 Interrupt for successful address/ data communication.
  - 1 Interrupt for error condition.
- Optional clock stretching.
- 1-byte buffer with DMA capability.
- Configurable PEC (packet error checking) generation or verification:
  - PEC value can be transmitted as last byte in Tx mode.
  - PEC error checking for last received byte.
  - SMBus 2.0 Compatibility:
  - 25 ms clock low timeout delay.
  - 10 ms master cumulative clock low extend time.
  - 25 ms slave cumulative clock low extend time.
  - Hardware PEC generation/verification with ACK control.
  - Address Resolution Protocol (ARP) supported.
- PMBus Compatibility.

1.5.8. SysTick

1.5.8.1. Introduction

The processor has a 24-bit system timer, SysTick, that counts down from the reload value to zero, reloads (wraps to) the value in the LOAD register on the next clock edge, then counts down on subsequent clocks. When the processor is halted for debugging the counter does not decrement.

1.5.8.2. General Features

The SysTick_config function automatically starts the System Timer, sets the countdown value as its only input, and enables the SysTick interrupt. By dividing the core frequency by 1,000, we are saying that we want each SysTick to occur every millisecond. For example, if the clock is running at 32MHz, the countdown value is 320,000. With that clock speed, that number of counts occur every 1ms. If the clock is running at 14MHz, the countdown value is 140,000, which also happens every 1ms.

1.5.9. Timer

1.5.9.1. General Purpose timers' introduction

The general-purpose timers consist of a 16-bit auto-reload counter driven by a programmable prescaler. They may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture) or generating output waveforms (output compare and PWM). Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

1.5.9.2. General-purpose timers' features

- 16-bit up, down, up/down auto-reload counter.
- 16-bit programmable prescaler used to divide (also "on the fly") the counter clock frequency by any factor between 1 and 65536.
- Up to 4 independent channels for:
    - Input capture.
    - Output compare.
    - PWM generation (Edge- and Center-aligned modes).
    - One-pulse mode output.
- Synchronization circuit to control the timer with external signals and to interconnect several timers.
- Interrupt/DMA generation on the following events:
    - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger).
    - Trigger event (counter start, stop, initialization or count by internal/external trigger).
    - Input capture.
    - Output compare.
- Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes.
- Trigger input for external clock or cycle-by-cycle current management

1.5.10. NVIC

1.5.10.1. NVIC Introduction

The NVIC provides configurable interrupt handling abilities to the processor. It facilitates low-latency exception and interrupt handling and controls power management.

1.5.10.2. NVIC Main Features

- up to 81 interrupts (depends on the STM32 device type).
- A programmable priority level of 0-15 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Level and pulse detection of interrupt signals.
- Dynamic reprioritization of interrupts.
- Grouping of priority values into group priority and subpriority fields.
- Interrupt tail-chaining.
- An external non-maskable interrupt (NMI).

1.5.11. CRC32

1.5.11.1. Introduction

The CRC (cyclic redundancy check) calculation unit is used to get a CRC code from a 32-bit data word and a fixed generator polynomial. Among other applications, CRC-based techniques are used to verify data transmission or storage integrity. In the scope of the EN/IEC 60335-1 standard, they offer a means of verifying the Flash memory integrity. The CRC calculation unit helps compute a signature of the software during runtime, to be compared with a reference signature generated at link time and stored at a given memory location.

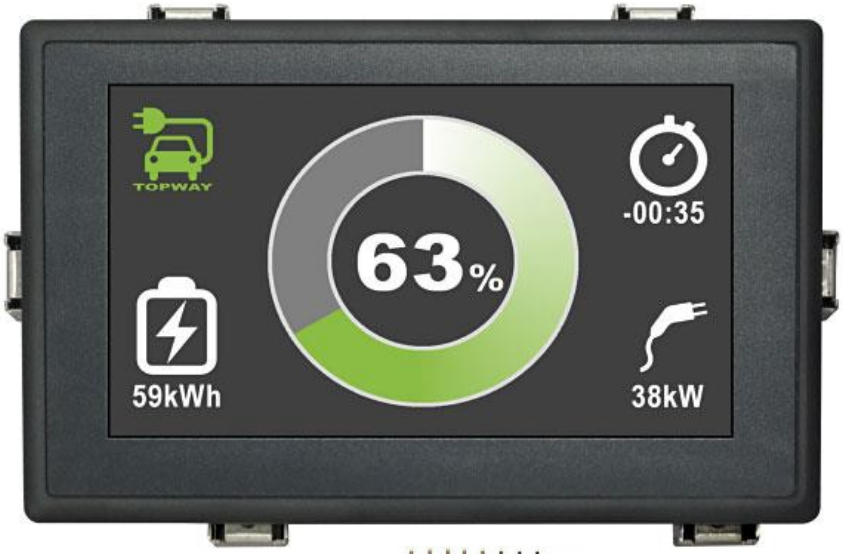
1.5.11.2. CRC32 Main Features

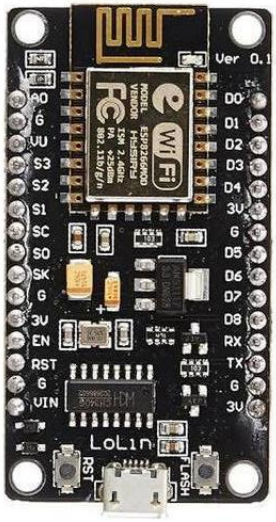
- Uses CRC-32 (Ethernet) polynomial: 0x4C11DB7A programmable –X 32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 +X8 + X7 + X5 + X4 + X2+ X + 1.
- Single input/output 32-bit data register.
- CRC computation done in 4 AHB clock cycles (HCLK).
- General-purpose 8-bit register (can be used for temporary storage).

```
uint32_t Compute_CRC32_HEX(uint8_t HEX_byte)
{
  const uint32_t polynomial = 0x04C11DB7; // divisor 32bit //
  uint8_t b = HEX_byte;
  crc ^= (uint)(b << 24); /* move byte into MSB of 32bit CRC */
  for (int i = 0; i < 8; i++)
  {
    if ((crc & 0x80000000) != 0) /* test for MSB = bit 31 */
    {
      crc = (uint)((crc << 1) ^ polynomial);
```

```
      }
      Else
      {
        crc <<= 1;
      }
    }
    return crc;
  }
  void CRC32_MPEG_reset()
  {
    crc = 0xFFFFFFFF;
  }
```

1.6. Registration Device





The first part of our prototype is the registration device. Besides the main microcontroller it consists of a WiFi module ESP8266 acting as the gateway to the local server and hence to the cloud, RFID reader used to scan the patient's card and identify him or assign new card for new patient. Throughout this chapter we'll take a quick look on the software architecture of the application, peripherals and how they are interfaced.

1.6.1. System Requirements

The registration device should be able to:

- employee login, to identify employee identity.
- Read RFID Tag, identify patient's identity, and report error if not found.
- card assigning for new patient by writing a new id on new card.
- Send id to local server to know if is exist or not.
- If patient id is not found or card is corrupted inform employee to assign a new card for patient by displaying a message in TFT display.
- Display Patient Data if needed.
- Ids are sent from STM32F103C8 to the local server via ESP8266 Wi-Fi module.
- Communication between STM32F103C8 and ESP8266 is held using UART.

1.6.2. Physical Layout

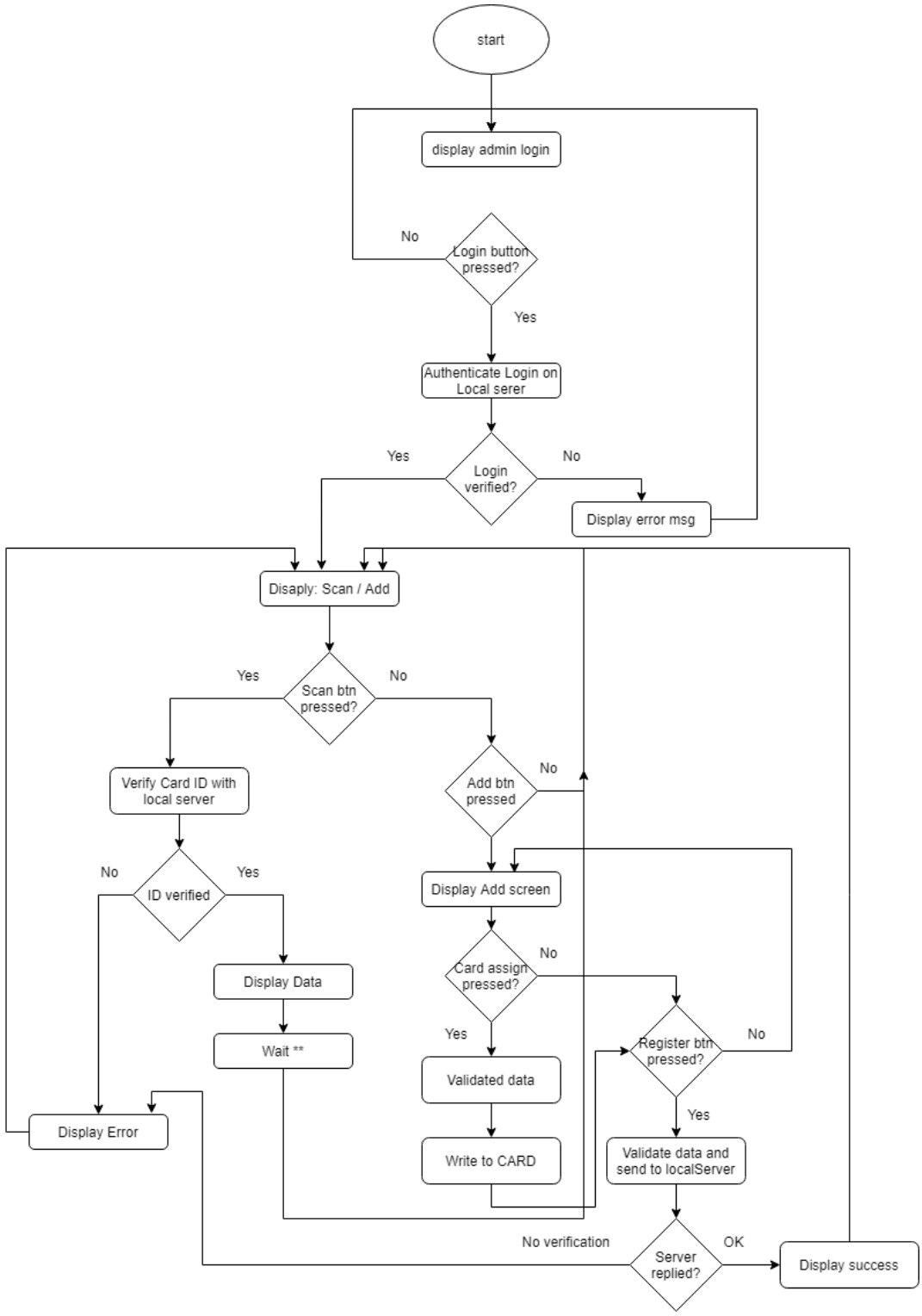
STM32F103C8	RFID reader
A5	Clk
A6	MISO
A7	MOSI
A4	SDA
A3	RST

STM32F103C8	ESP8266 (NODE MCU)
A9 (Tx)	D7 (Rx)
A10 (Rx)	D8 (Tx)

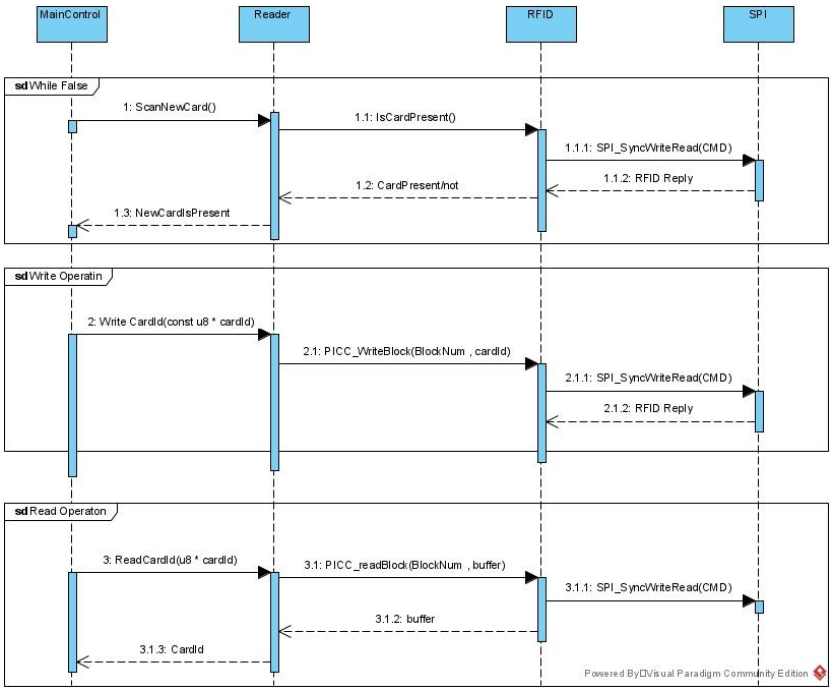
STM32F103C8	TFT HTK070DTA-1C
B10	TX
B11	RX

1.6.3. Software Architecture

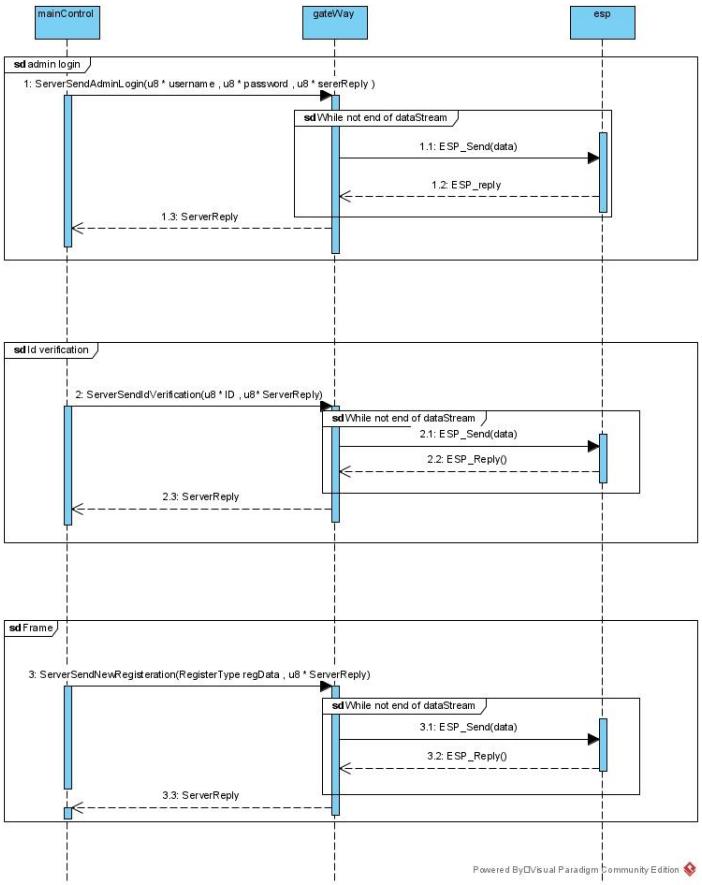
In this section, with the help of various UML diagrams- we'll get to understand how the architecture of the application is designed. The overall working of the system can be summarized as shown in the flowchart in figure 6.1. At start up, employee must enter his login data and wait to be verified then system initializes its components and go in a semi-idle state where only the reader task is polling the presence of a card and in sleep mode the rest of the time. Once a new card is found, data is read and verified at the local server or the cloud.



Another useful way to represent the system is using the state diagrams. Since it simply shows task flow and interaction between system components.



Reader Sequence Diagram



Gateway Sequence Diagram

1.6.4. Device Peripherals

1.6.4.1. TFT HKT070DTA-1C Driver

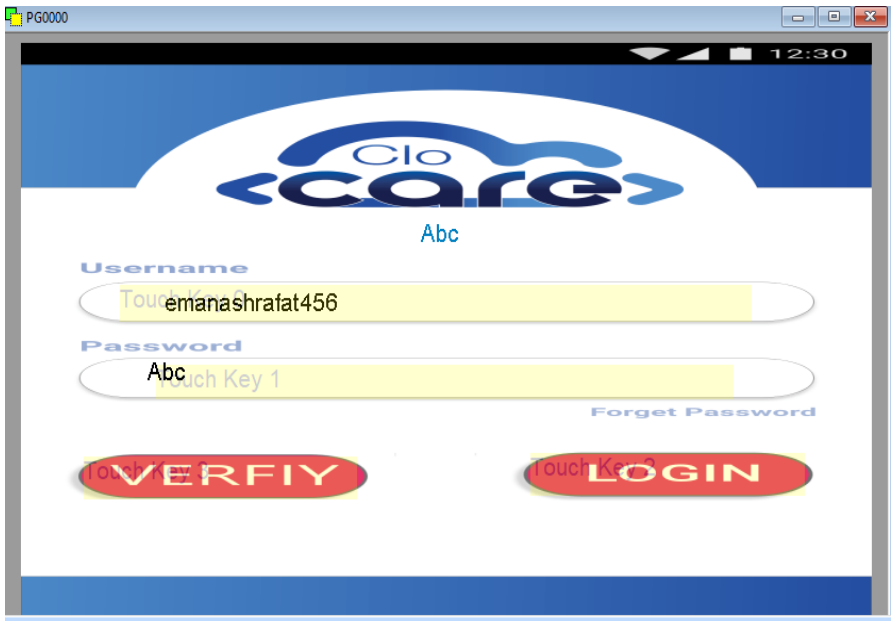
Smart TFT Module with 32bit MCU on broad. Its graphics engine provides number of outstanding features .it supports topway TML 3 For preload and pre-design display interface that simplify the operation and development time. Suitable for industry control instrument, medical electronics ,power electric equipment applications.

1.6.4.2. Reader Driver

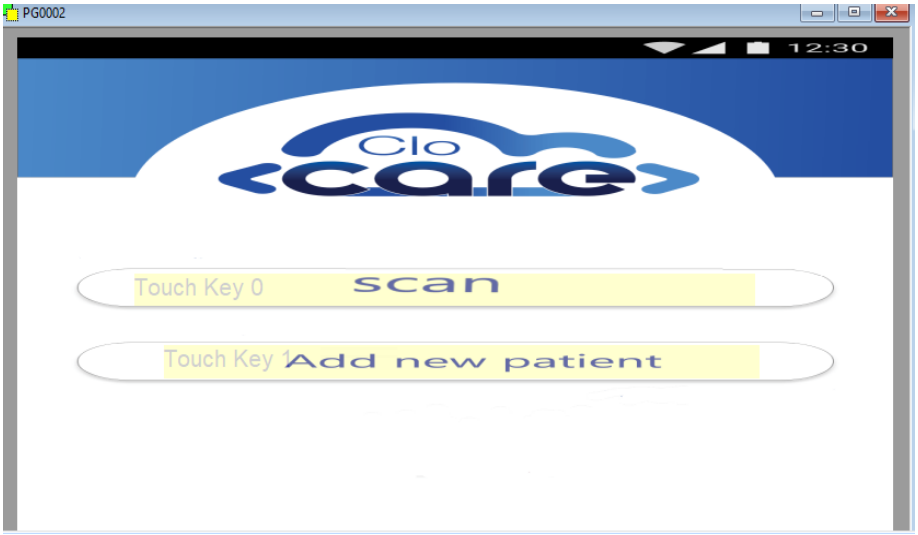
To read a card data first, you need to initialize RFID module and then keep polling for a new card scan, if found authenticate card and read its data. Gateway ESP8266 Communication using UART modules is used to send login data and employee login data.

1.6.4.3. UI Design for TFT Display

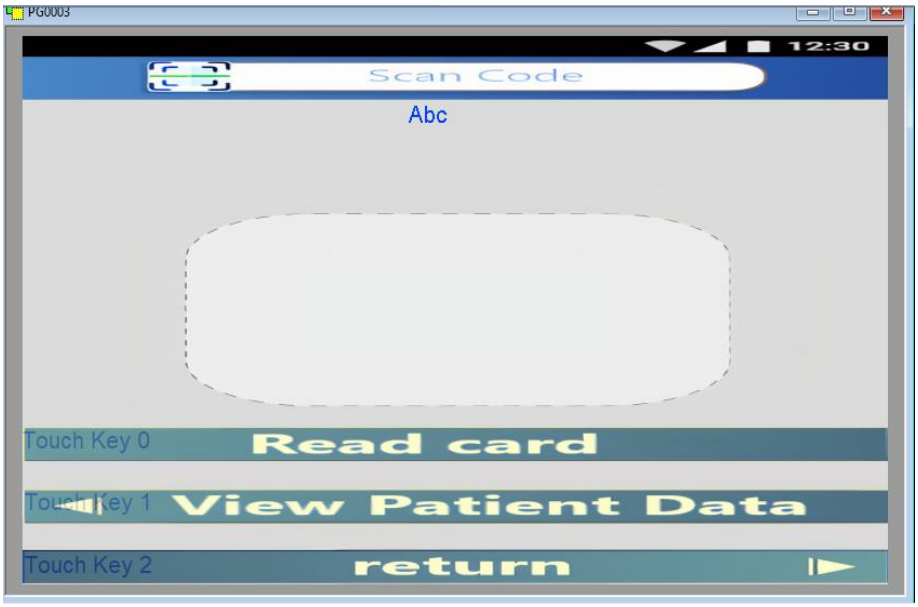
Employee login:



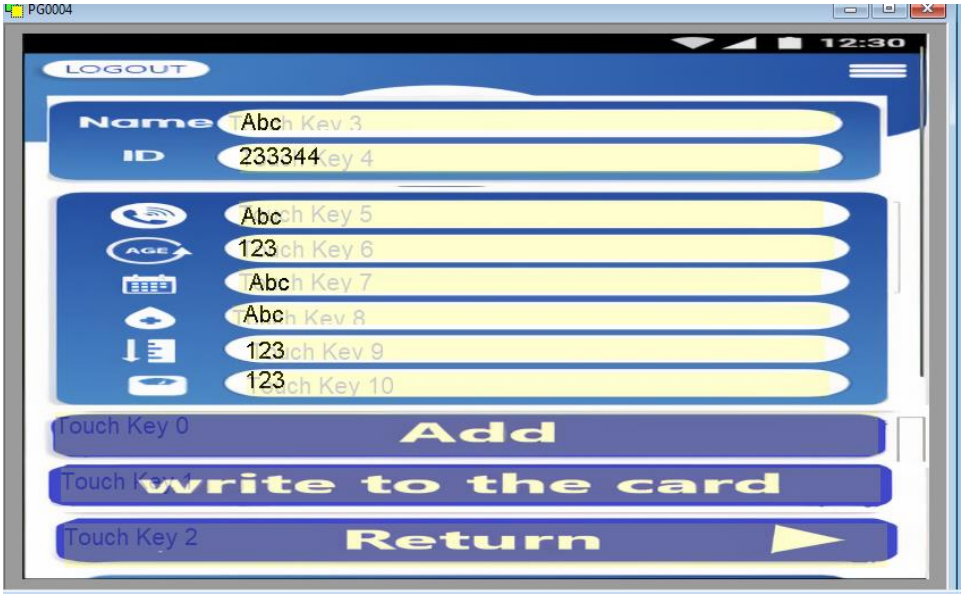
Scan/ add patient select:



Scan card

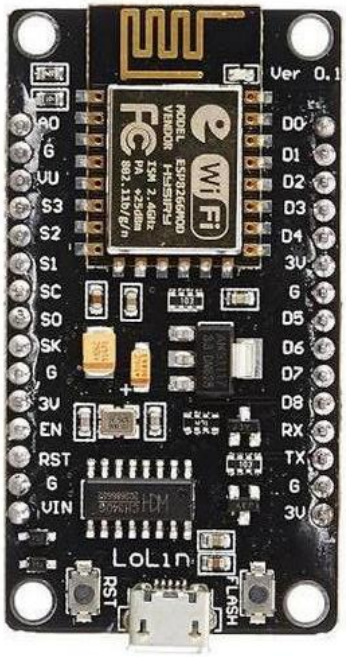


Add patient:



Display patient Data:

1.7. Monitoring device



The second part of our prototype is the monitoring device. Besides the main microcontroller it consists of a WiFi module ESP8266 acting as the gateway to the local server and hence to the cloud, RFID reader used to scan the patient's card and identify him to know where and how to store his biological data, body temperature sensor and MAX30102 heartrate and SPO2 sensor for a proof-of-concept purposes, and finally a TFT display to represent a visual output to the world. Throughout this chapter we'll take a quick look on the software architecture of the application, peripherals and how they are interfaced, and we'll get a peek on the real-time operating system used to implement the functionality concurrently.

1.7.1. System Requirements

The monitoring device should be able to:

- Read RFID Tag, identify patient's identity, and report error if not found.
- Read, log, verify, and display sensor's reading only if a card is registered.
- Require card assigning if monitoring sensors are detached within 1 min cooldown period. i.e., sensors reading are out of valid range.
- Send periodic data readings to local server only if a card is assigned to the device along with patient token.
- Send an emergency message if sensors' readings are abnormal (in critical state).
- Tags, display, and Sensors' processing is performed on the STM32F103C8 microcontroller.
- Messages are sent from STM32F103C8 to the local server via ESP8266 Wi-Fi module.
- Communication between STM32F103C8 and ESP8266 is held using UART.

1.7.2. Physical Layout

Pinout mapping of the different peripherals with respect to the main microcontroller is as follows:

STM32F103C8	RFID reader
A5	Clk
A6	MISO
A7	MOSI
A4	SDA
A3	RST

STM32F103C8	TFT ST7735
B13	CLK
B14	MISO
B15	MOSI
B12	CS
B3	DC
B4	RST
B5	LED

STM32F103C8	ESP8266 (NODE MCU)
A9 (Tx)	D7 (Rx)
A10 (Rx)	D8 (Tx)

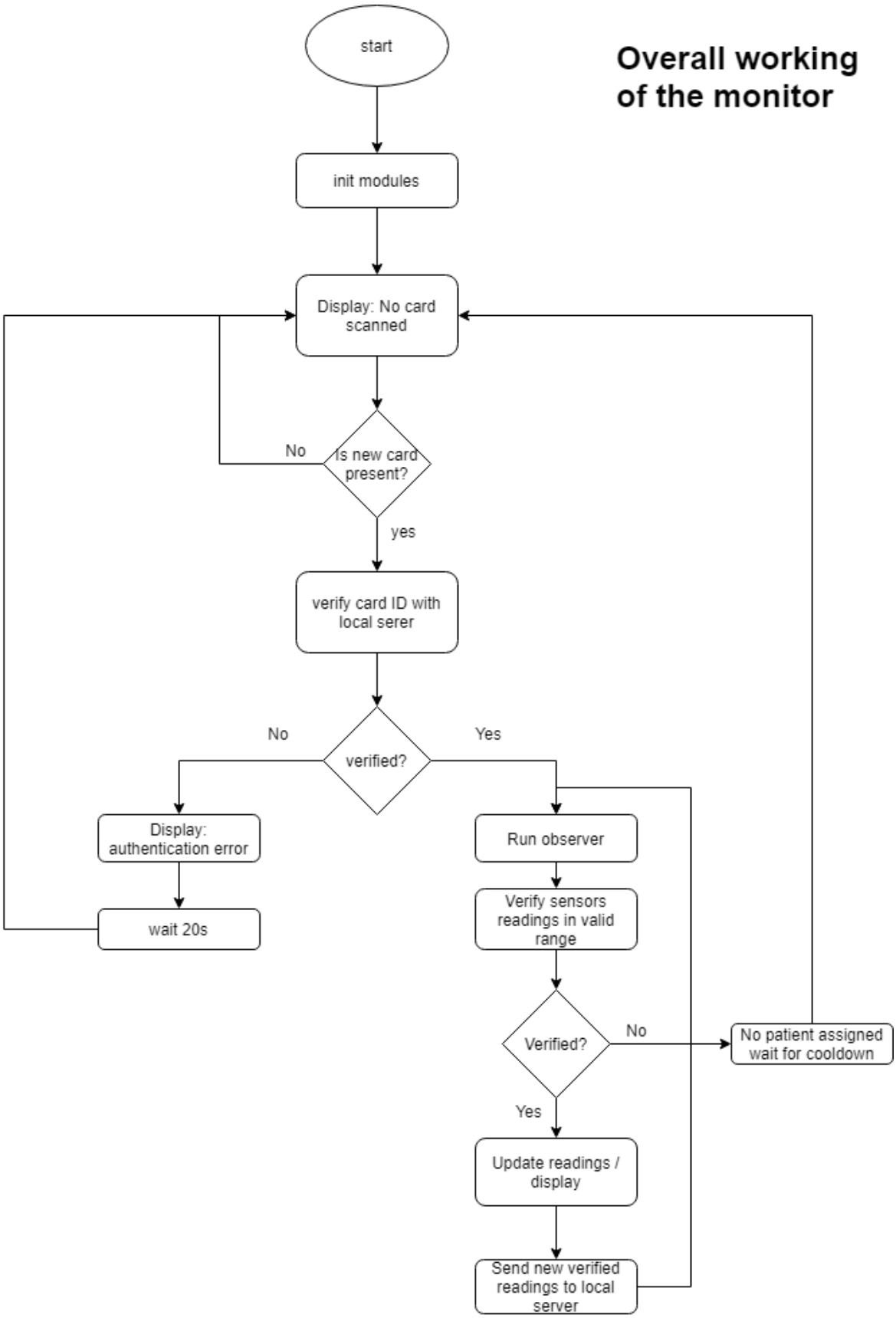
STM32F103C8	SPO2 module
B6	SCL
B7	SDA

STM32F103C8	Body Temp sensor
B0, B1	ADC PIN

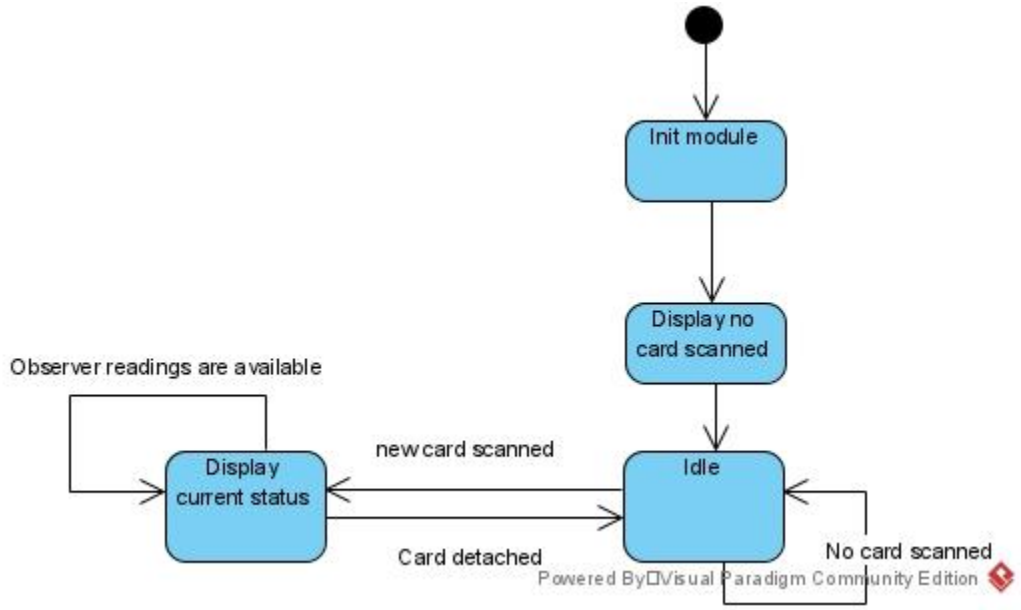
1.7.3. Software Architecture

In this section, -with the help of various UML diagrams- we'll get to understand how the architecture of the application is designed.

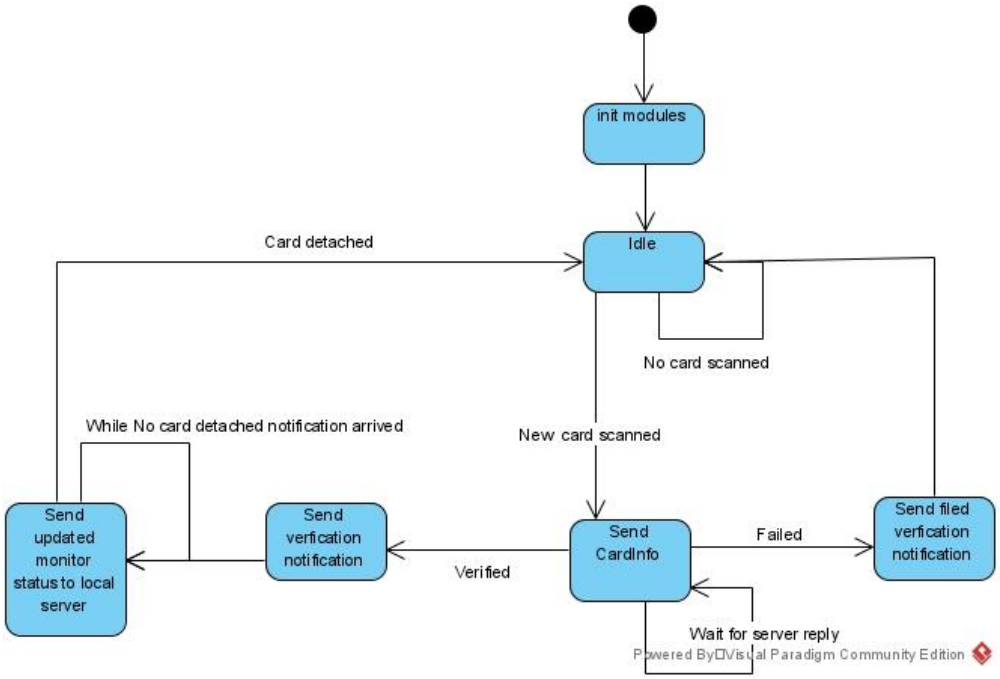
The overall working of the system can be summarized as shown in the flowchart in figure 7.1. At start up, the system initializes its components, and go in a semi-idle state where only the reader task is polling the presence of a card and in sleep mode the rest of the time. Once a new card is found, login data is read and verified at the local server or the cloud, and then the monitor can start performing sensors readings and validation. If all sensors give invalid readings for a fixed amount of time, the system shall reset and consider the patient detached from the monitor.


Overall working of the monitor
start
init modules
Display: No card scanned
Is new card present?
No
yes
verify card ID with local serer
verified?
No
Yes
Display: authentication error
wait 20s
Run observer
Verify sensors readings in valid range
Verified?
No
No patient assigned wait for cooldown
Yes
Update readings / display
Send new verified readings to local server

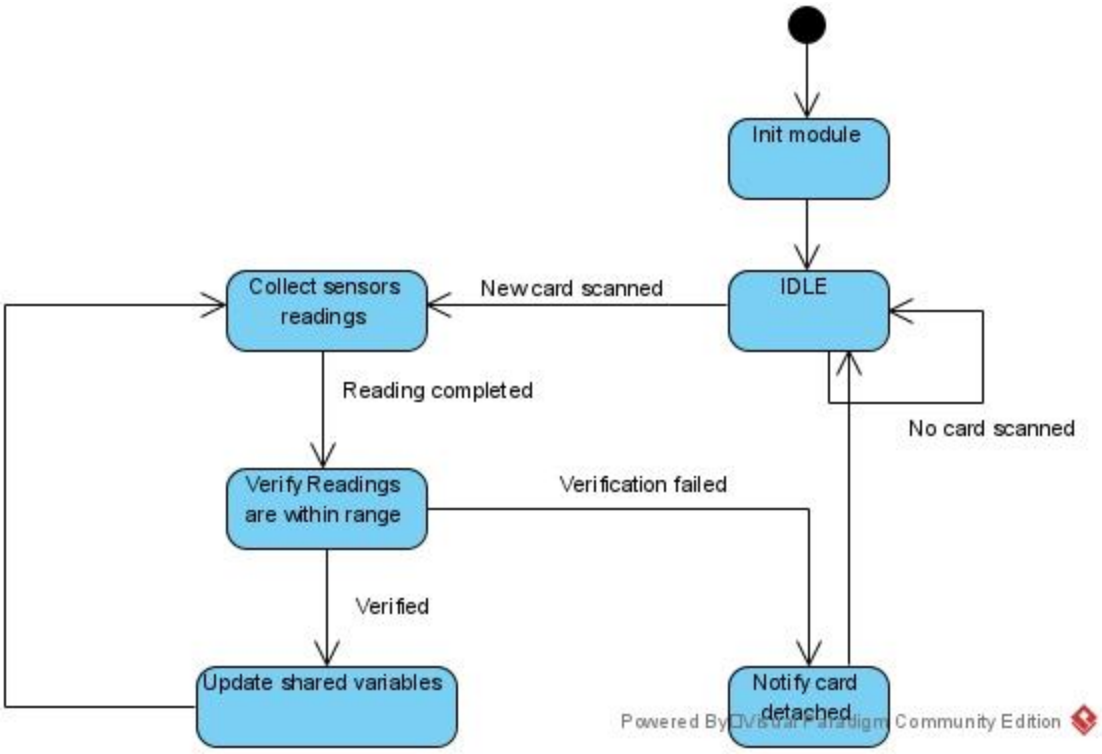
Another useful way to represent the system is using the finite state machine diagrams. Since the application is divided into a set of tasks, each task can have its FSM.



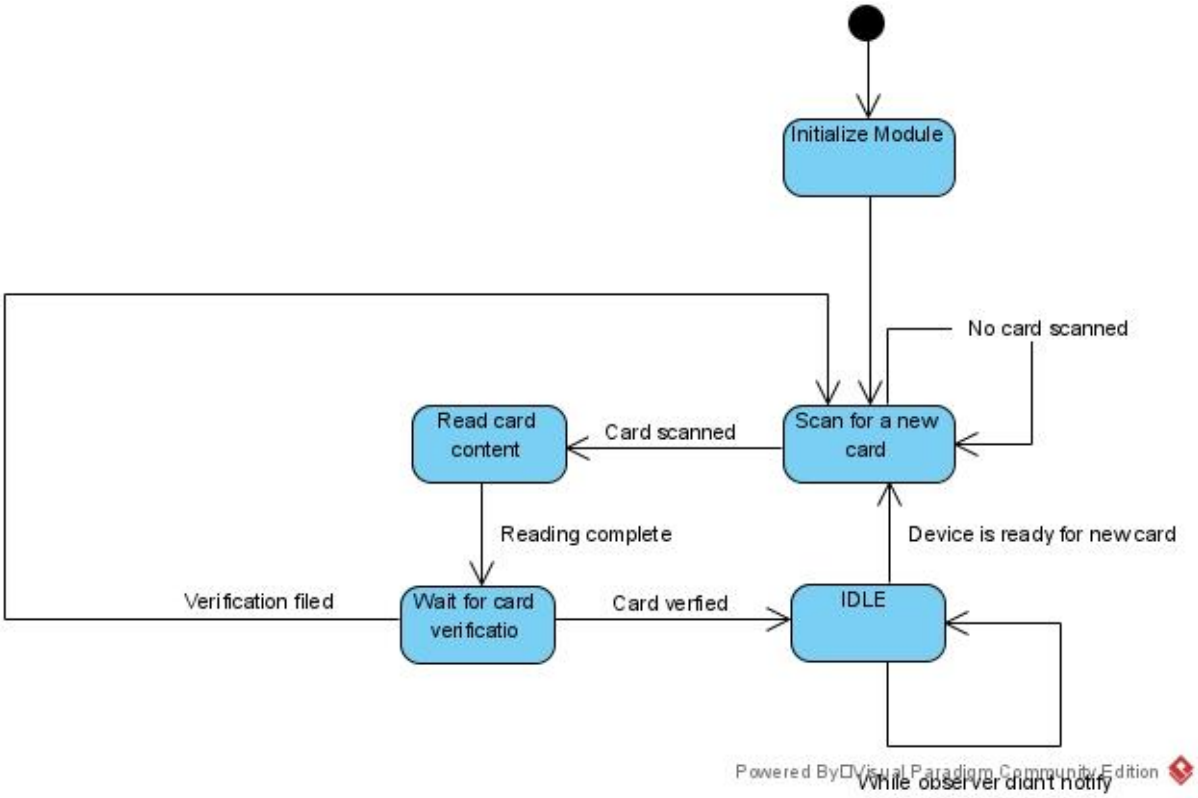
Display FSM



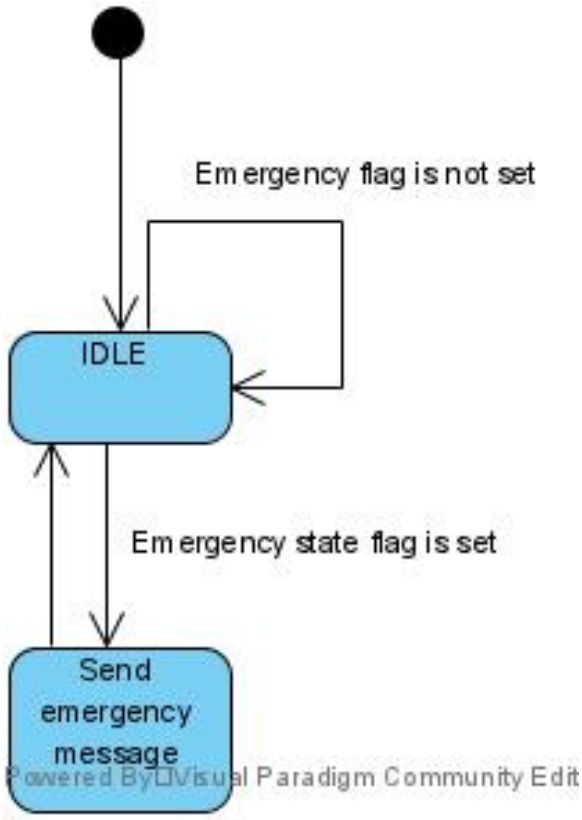
Gateway FSM



Observer FSM



Reader FSM



Emergency FSM

1.7.4. Device Peripherals

The monitor device is responsible for reading card data, collecting sensors readings, validate readings, and assign them to patient data on the local server via WiFi module.

1.7.5. RFID reader

To read a card data first, you need to initialize RFID module and then keep polling for a new card scan, if found authenticate card and read its data. This functionality is implemented as follows:

```
// 1- include Libraries
#include "STD_TYPES.h"
#include "BIT_MATH.h"

// 2- include interface file of needed lower layers
#include "mfrc522.h"

// 3- include interface files
#include "Reader.h"
```

```
#include "Observer.h"

MIFARE_Key key;

/*********************************************************************/
/*                        Local Functions                            */
/*********************************************************************/
static boolean Reader_AuthCard(uint8 block);
static boolean Reader_WriteLoginData(loginData_Type login);
static boolean Reader_ReadID(loginData_Type *login);
static boolean Reader_ReadPASS(loginData_Type *login);

/*********************************************************************/
/*                         APIs Definitions                          */
/*********************************************************************/

/****************************************
 * @name : Reader_init
 * @param:
 * Non Reentrant
 * Sync
 * Service to init Reader.
 ****************************************/
void Reader_init(void)
{
  // initialize RFID module
  PCD_Init();
  // initalize keys
  uint8 i;
  for (i = 0; i < 6; i++)
    key.keyByte[i] = 0xFF;

  return;
}

/****************************************
 * @name : Reader_isNewCardPresent
 * @param:
 * Non Reentrant
 * Sync
 * Service to query RFID reader.
 ****************************************/
boolean Reader_isNewCardPresent(void)
{
  boolean ret = FALSE;
```

```
  if (PICC_IsNewCardPresent())
  {
    PICC_ReadCardSerial();
    ret = TRUE;
  }
  return ret;
}

/*****************************************
 * @name : Reader_GetId
 * @param:
 * @return: Id read from card if success or zero otherwise.
 * Non Reentrant
 * Sync
 * Service to read card ID.
 *****************************************/
loginData_Type Reader_GetLogin(void)
{
  loginData_Type ID;
  volatile loginData_Type ret_ID = {0};
  uint8 i = 0;
  if (Reader_AuthCard(ID_BLOCK) == TRUE)
  {
    if (Reader_ReadID(&ID) == TRUE)
    {
      for (i = 0; i < ID_READ_LEN; i++)
        ret_ID.id_r[i] = ID.id_r[i];
    }
  }
  if (Reader_AuthCard(PASS_BLOCK) == TRUE)
  {
    if (Reader_ReadPASS(&ID) == TRUE)
    {
      for (i = 0; i < PASS_READ_LEN; i++)
        ret_ID.Pass_r[i] = ID.Pass_r[i];
    }
  }
  __asm("NOP");
  return ret_ID;
}

/*************************************************************************/
/*                     Local Functions definitions                       */
/*************************************************************************/
static boolean Reader_AuthCard(uint8 block)
```

```
{
  volatile boolean ret = FALSE;
  Uid uid = PCD_getUid();
  if (PCD_Authenticate(PICC_CMD_MF_AUTH_KEY_A, block, &key, &uid) == mfrc522_STAT
US_OK)
  {
    ret = TRUE;
  }
  return ret;
}

static boolean Reader_ReadID(loginData_Type *ID)
{
  boolean ret = FALSE;
  int status = PICC_ReadBlock(ID_BLOCK, ID->id_r, &key);
  if (status == mfrc522_STATUS_OK)
  {
    ret = TRUE;
  }
  return ret;
}

static boolean Reader_ReadPASS(loginData_Type *login)
{
  boolean ret = FALSE;
  int status = PICC_ReadBlock(PASS_BLOCK, login->Pass_r, &key);
  if (status == mfrc522_STATUS_OK)
  {
    ret = TRUE;
  }
  return ret;
}
```

1.7.6. SPO2 driver

The spo2 main function is polling the sensor to see if it has any new samples and buffer them to the application. The main function used to implement this functionality is as following:

```
uint16 MAX30102_check(void)
{
  //Read register FIDO_DATA in (3-byte * number of active LED) chunks
  //Until FIFO_RD_PTR = FIFO_WR_PTR
```

```
  uint8 readPointer = MAX30102_getReadPointer();
  uint8 writePointer = MAX30102_getWritePointer();

  sint16 numberOfSamples = 0;

  //Do we have new data?
  if (readPointer != writePointer)
  {
    //Calculate the number of readings we need to get from sensor
    numberOfSamples = writePointer - readPointer;
    if (numberOfSamples < 0)
      numberOfSamples += 32; //Wrap condition

    //We now have the number of readings, now calc bytes to read
    //For this example we are just doing Red and IR (3 bytes each)
    sint16 bytesLeftToRead = numberOfSamples * activeLEDs * 3;

    //Get ready to read a burst of data from the FIFO register
    uint8 reg = MAX30102_FIFO_DATA;
    I2C_voidMasterSendData(I2C1, &reg, 1, MAX30102_SLAVE_ADDRESS, I2C_NO_REPEAT_S
);

    //We may need to read as many as 288 bytes so we read in blocks no larger tha
n I2C_BUFFER_LENGTH
    while (bytesLeftToRead > 0)
    {
      sint16 toGet = bytesLeftToRead;
      if (toGet > I2C_BUFFER_LENGTH)
      {
        //If toGet is 32 this is bad because we read 6 bytes (Red+IR * 3 = 6) at
a time
        //32 % 6 = 2 left over. We don't want to request 32 bytes, we want to req
uest 30.
        //32 % 9 (Red+IR+GREEN) = 5 left over. We want to request 27.

        //Trim toGet to be a multiple of the samples we need to read
        toGet = I2C_BUFFER_LENGTH - (I2C_BUFFER_LENGTH % (activeLEDs * 3));
      }

      bytesLeftToRead -= toGet;

      //Request toGet number of bytes from sensor
      uint8 recBuffer[MAX_BUFFER];
      uint8 toGetIdx = 0;
      I2C_voidMasterReceiveData(I2C1, recBuffer, toGet, MAX30102_SLAVE_ADDRESS);
```

```
      while (toGet > 0)
      {
        sense.head++;                //Advance the head of the storage struct
        sense.head %= STORAGE_SIZE; //Wrap condition

        uint8 temp[sizeof(uint32)] = {0}; //Array of 4 bytes that we will convert
 into long
        uint32 tempLong;
        //Burst read three bytes - RED

        temp[3] = 0;
        temp[2] = recBuffer[toGetIdx++];
        temp[1] = recBuffer[toGetIdx++];
        temp[0] = recBuffer[toGetIdx++];

        //Convert array to long
        tempLong = (uint32)(temp[0] << 0) | (uint32)(temp[1] << 8) | (uint32)(tem
p[2] << 16);
        tempLong &= 0x3FFFF;              //Zero out all but 18 bits
        sense.red[sense.head] = tempLong; //Store this reading into the sense arr
ay

        if (activeLEDs > 1)
        {

          temp[3] = 0;
          temp[2] = recBuffer[toGetIdx++];
          temp[1] = recBuffer[toGetIdx++];
          temp[0] = recBuffer[toGetIdx++];
          //Convert array to long
          tempLong = (uint32)(temp[0] << 0) | (uint32)(temp[1] << 8) | (uint32)(t
emp[2] << 16);
          tempLong &= 0x3FFFF; //Zero out all but 18 bits
          sense.IR[sense.head] = tempLong;
        }

        toGet -= activeLEDs * 3;
      }

    } //End while (bytesLeftToRead > 0)

  } //End readPtr != writePtr

  return (numberOfSamples); //Let the world know how much new data we found }
```

1.7.7. Temp sensor

Using 12-bits ADC the temperature sensors reading is gathered and converted to a temperature following the specified equations of the data sheet as following:

```
float32 TempSensor_f32TempRead(void)
{
  uint16 Loc_u16val = 0;
  float32 Loc_f32celsius = 0.0;

  /* reading ADC1 value */
  Loc_u16val = Read_ADC();

  /* calculating R */
  /* (65535/ADC - 1) */
  Loc_u16val = (4096 / Loc_u16val) - 1;

  /* 10K / (65535/ADC - 1) */
  Loc_u16val = SERIESRESISTOR / Loc_u16val;

  /* calculating T */
  /* (R/Ro) */
  Loc_f32celsius = (float32)1.0 * Loc_u16val / THERMISTORNOMINAL; /* ln(R/Ro) */
  Loc_f32celsius = log(Loc_f32celsius);
  /* 1/B * ln(R/Ro) */
  Loc_f32celsius /= BCOEFFICIENT;
  Loc_f32celsius += (float32)1.0 / (float32)(TEMPERATURENOMINAL + 273.15); /* + (
1/To) */
  Loc_f32celsius = (float32)1.0 / (float32)Loc_f32celsius;                /* Inv
ert */
                                                                          /* con
vert K temp to C */
  Loc_f32celsius -= (float32)273.15;
  return (Loc_f32celsius);
}
```

1.7.8. Gateway ESP8266

Communication using UART modules is used to send login data and sensors' readings and is implemented as following:

```
{
  ESP_init();
  // Send Hi msg
  //ESP_writeSync("Hello.\n");
  return;
```

```
}

boolean GateWay_WriteSensorsReadings(const ObserverReadingsType readings)
{
  // Send cmd first
  uint8 *cmd = "READS\n";
  ESP_writeSync(cmd);
  uint8 reply[20];
  boolean ret = FALSE;
  // Construct string of data
  uint8 Readings[30] = "";
  AppendFloatToString(Readings, readings.Temp);
  AppendChar(Readings, '\n');
  AppendIntegerToString(Readings, readings.SPO2);
  AppendChar(Readings, '\n');
  AppendIntegerToString(Readings, readings.HeartRate);
  AppendChar(Readings, '\n');

  // Send data
  ESP_writeSync(Readings);
  ESP_readSync(reply, 4);
  if (reply[0] == 'O' && reply[1] == 'K')
  {
    ret = TRUE;
  }
  return ret;
}

boolean GateWay_WriteLoginData(const loginData_Type loginData)
{
  uint8 *cmd = "\nLOGIN_PA\n";
  uint8 national_id[14];
  uint8 passWord[14];
  uint8 i = 0;
  uint8 reply[20];
  boolean ret = FALSE;
  for (i = 0; i < 14; i++)
  {
    national_id[i] = loginData.id_r[i];
    passWord[i] = loginData.Pass_r[i];
  }

  ESP_writeSync(cmd);
  ESP_writeSync(national_id);
  ESP_writeSync("\n");
```

```
  ESP_writeSync(passWord);
  ESP_writeSync("\n");
  ESP_readSync(reply, 4);

  if (reply[0] == 'O' && reply[1] == 'K')
  {
    ret = TRUE;
  }
  return ret;
}

void GateWay_SendEmergencyMsg(void)
{
  uint8 *msg = "EMG\n";
  ESP_writeSync(msg);
  return;
}
```

1.7.9. Real Time Operating system

The monitoring device handles several tasks at the same time. Converting ADC results, reading i2c buffer, displaying, and sending UART messages. For efficient handling of these operations, a real time operating system is very helpful to use.

1.7.9.1. Multi-Threading

Multithreading is the ability of a program or an operating system process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer.

1.7.9.2. RTOS overview

A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time applications that process data as it comes in, typically without buffer delays. Processing time requirements (including any OS delay) are measured in tenths of seconds or shorter increments of time. A real-time system is a time-bound system which has well-defined, fixed time constraints. Processing must be done within the defined constraints, or the system will fail. They either are event-driven or time-sharing. Event-driven systems switch between tasks based on their priorities, while time-sharing systems switch the task based on clock interrupts. Most RTOSs use a pre-emptive scheduling algorithm.

1.7.9.3. Free-RTOS specifications

FreeRTOS is designed to be small and simple. The kernel itself consists of only three C files. To make the code readable, easy to port, and maintainable, it is written mostly in C, but there are a few assembly functions included where needed (mostly in architecture-specific scheduler routines).

FreeRTOS provides methods for multiple threads or tasks, mutexes, semaphores and software timers. A tick-less mode is provided for low power applications. Thread priorities are supported. FreeRTOS applications can be completely statically allocated. Alternatively RTOS objects can be dynamically allocated with five schemes of memory allocation provided:

- allocate only.
- allocate and free with a very simple, fast, algorithm.
- a more complex but fast allocate and free algorithm with memory coalescence.

an alternative to the more complex scheme that includes memory coalescence that allows a heap to be broken across multiple memory areas. and C library allocate and free with some mutual exclusion protection.

There are none of the more advanced features typically found in operating systems like Linux or Microsoft Windows, such as device drivers, advanced memory management, user accounts, and networking. The emphasis is on compactness and speed of execution. FreeRTOS can be thought of as a 'thread library' rather than an 'operating system', although command line interface and POSIX-like I/O abstraction add-ons are available.

FreeRTOS implements multiple threads by having the host program call a thread tick method at regular short intervals. The thread tick method switches tasks depending on priority and a round-robin scheduling scheme. The usual interval is 1 to 10 milliseconds (1/1000 to 1/100 of a second), via an interrupt from a hardware timer, but this interval is often changed to suit a particular application.

The download contains prepared configurations and demonstrations for every port and compiler, allowing rapid application design. The FreeRTOS.org site also contains a lot of documentation and RTOS tutorials (additional manuals and tutorials are available for a fee), as well as details of the RTOS design.

Key features:

- Book and reference manuals.
- Small memory footprint, low overhead, and fast execution.
- Tick-less option for low power applications.
- Intended for both hobbyists and professional developers working on commercial products.
- Scheduler can be configured for both preemptive or cooperative operation.
- Coroutine support (coroutines in FreeRTOS are simple and lightweight tasks with limited use of the call stack)

1.7.10. OS tasks

Now that we understand how RTOS works, we split up the system to a set of tasks running simultaneously.

1.7.10.1. Reader Task

```
void Reader_MainTask(void *pvParameters)
{
  loginData_Type ID = {"2981001234908", "", "2981001234908", ""};
  uint32_t NotifiedVal = 0;

  while (1)
  {
    //NotifiedVal = 0x00;
    if (Reader_isNewCardPresent() == TRUE)
    {
      // Get ID to be used by the GateWay task in local server communication
      //  Reader_SetLoginData(ID);
      ID = Reader_GetLogin();
      xQueueSend(xCardIdQueue, &ID, portMAX_DELAY);

      // Notify Observer, Display, Gateway, and emergency task to start working
      xTaskNotify(DisplayHandle, NEW_CARD_PRESENT_NOTIFICATION, eSetValueWithOver
write);
      xTaskNotify(GateWayHandle, NEW_CARD_PRESENT_NOTIFICATION, eSetValueWithOver
write);
      vTaskDelay(20 / portTICK_PERIOD_MS); // Delay to enable Login process to co
mplete.
```

```
      xTaskNotify(Spo2Handle, NEW_CARD_PRESENT_NOTIFICATION, eSetValueWithOverwri
te);
      xTaskNotify(ObserverHandle, NEW_CARD_PRESENT_NOTIFICATION, eSetValueWithOve
rwrite);

      // Go Idle and wait for card detached notification
      xTaskNotifyWait(0xffffffff, 0x00, &NotifiedVal, portMAX_DELAY);
      Reader_init();
    }
    else
    {
      vTaskDelay(100 / portTICK_PERIOD_MS);
    }
  }
}
```

1.7.10.2. Observer Task

```
void Observer_mainTask(void *pvParameters)
{
  uint32_t NotifiedVal;
  ObserverReadingsType readings = {0};
  volatile uint8 err = 0;
  volatile uint8 err_counter = 0;

  while (1)
  {
    if ((NotifiedVal & NEW_CARD_PRESENT_NOTIFICATION) == NEW_CARD_PRESENT_NOTIFIC
ATION)
    {
      err = Observer_GetCurrentTemp(&readings);
      if (xQueueReceive(Spo2Queue, &(readings.SPO2), 100) == pdFALSE)
      {
        err++;
      }
      else if ((readings.SPO2 < SPO2_MIN) || (readings.SPO2 > SPO2_MAX))
      {
        err++;
      }
      else
      {
        //ok
      }
```

```
      if (xQueueReceive(HRQueue, &(readings.HeartRate), 100) == pdFALSE)
      {
        err++;
      }
      else if (readings.HeartRate < HEARTRATE_MIN || readings.HeartRate > HEARTRA
TE_MAX)
      {
        err++;
      }
      else
      {
        //ok
      }
      xQueueOverwrite(xSensorsDataumQueue, &readings);
      /* Check if values are in critical range while still in valid readings rang
e */
      if ((readings.HeartRate <= HEARTRATE_CRITICAL_MIN && readings.HeartRate > H
EARTRATE_MIN) || (readings.SPO2 <= SPO2_CRITICAL_MIN && readings.SPO2 > SPO2_MIN)
 || (readings.Temp >= TEMP_CRITICAL_MIN && readings.Temp < TEMP_MAX))
      {
        xTaskNotify(EmergencyHandle, EMERGENCY_MSG_NOTIFICATION, eSetValueWithOve
rwrite);
        xTaskNotify(DisplayHandle, EMERGENCY_MSG_NOTIFICATION, eSetValueWithOverw
rite);
        vTaskDelay(10 / portTICK_PERIOD_MS); // to make sure notification is reci
eved.
      }
      else
      {
        // ok
      }

      /* check if all values are invalid */
      if (err > 2)
      {
        err_counter++;
      }
      else
      {
        err_counter = 0;
      }

      if (err_counter > MAX_ERROR_COUNT) // check if all values are invalid for x
 times in raw.
      {
```

```
        // No patient is assigned , scan a new card.
        xTaskNotify(DisplayHandle, CARD_DETACHED_NOTIFICATION, eSetValueWithOverw
rite);
        xTaskNotify(ReaderHandle, CARD_DETACHED_NOTIFICATION, eSetValueWithOverwr
ite);
        xTaskNotify(GateWayHandle, CARD_DETACHED_NOTIFICATION, eSetValueWithOverw
rite);
        xTaskNotify(Spo2Handle, CARD_DETACHED_NOTIFICATION, eSetValueWithOverwrit
e);

        // Go Idle and wait for Wake up notification
        xTaskNotifyWait(0xffffffff, 0x00, &NotifiedVal, portMAX_DELAY);
      }
      else if (err == 0)
      {
        // Update readings
        xTaskNotify(GateWayHandle, CARD_IS_PRESENT_NOTIFICATION, eSetValueWithOve
rwrite);
        // Update display.
        xTaskNotify(DisplayHandle, CARD_IS_PRESENT_NOTIFICATION, eSetValueWithOve
rwrite);
      }
      else
      {
        // Update display.
        xTaskNotify(DisplayHandle, CARD_IS_PRESENT_NOTIFICATION, eSetValueWithOve
rwrite);
      }

      vTaskDelay(200 / portTICK_PERIOD_MS); // Wait before new reading collection
    }
    else
    {
      xTaskNotifyWait(0xffffffff, 0x00, &NotifiedVal, portMAX_DELAY);
    }
  }
}
```

1.7.10.3. Display Task

```
void Display_mainTask(void *pvParameters)
```

```
{
  uint32_t NotifiedVal = 0;
  ObserverReadingsType reading;
  while (1)
  {

    if ((NotifiedVal & NEW_CARD_PRESENT_NOTIFICATION) == NEW_CARD_PRESENT_NOTIFIC
ATION)
    {
      Display_CurrentSensorsState();
      // vTaskDelay(500 / portTICK_PERIOD_MS);
    }
    else if (((NotifiedVal & CARD_IS_PRESENT_NOTIFICATION) == CARD_IS_PRESENT_NOT
IFICATION))
    {
      xQueuePeek(xSensorsDataumQueue, &(reading), portMAX_DELAY);
      Display_UpdateSensors(reading);
      Display_CurrentSensorsState();
    }
    else if ((NotifiedVal & CARD_DETACHED_NOTIFICATION) == CARD_DETACHED_NOTIFICA
TION)
    {
      Display_NoCard();
    }
    else if ((NotifiedVal & EMERGENCY_MSG_NOTIFICATION) == EMERGENCY_MSG_NOTIFICA
TION)
    {
      Display_Emeregency();
    }
    else
    {
    }
    xTaskNotifyWait(0xffffffff, 0x00, &NotifiedVal, portMAX_DELAY);
  }
}
```

1.7.10.4. Gateway Task

```
void GateWay_mainTask(void *pvParameters)
{
  uint32_t NotifiedVal = 0;
  ObserverReadingsType reading;
  loginData_Type login;
```

```
  boolean status = 0;
  while (1)
  {
    xTaskNotifyWait(0xffffffff, 0x00, &NotifiedVal, portMAX_DELAY);
    if ((NotifiedVal & NEW_CARD_PRESENT_NOTIFICATION) == NEW_CARD_PRESENT_NOTIFIC
ATION)
    {
      xQueuePeek(xCardIdQueue, &login, portMAX_DELAY);
      status = 0;
      while (status == 0)
      {
        // Keep try to log in
        status = GateWay_WriteLoginData(login);
      }
    }
    else if ((NotifiedVal & CARD_IS_PRESENT_NOTIFICATION) == CARD_IS_PRESENT_NOTI
FICATION)
    {
      xQueuePeek(xSensorsDataumQueue, &reading, portMAX_DELAY);
      GateWay_WriteSensorsReadings(reading);
      vTaskDelay(6000 / portTICK_PERIOD_MS); // Wait before sending again
    }
    else
    {
    }
  }
}
```

1.7.10.5. Emergency Task

```
void Emergency_mainTask(void *pvParameters)
{
  uint32_t NotifiedVal = 0;
  loginData_Type ID;
  while (1)
  {
    xTaskNotifyWait(0xffffffff, 0x00, &NotifiedVal, portMAX_DELAY);
    if ((NotifiedVal & EMERGENCY_MSG_NOTIFICATION) == EMERGENCY_MSG_NOTIFICATION)
    {
      xQueuePeek(xCardIdQueue, &ID, portMAX_DELAY);
      //      GateWay_WriteLoginData(ID);
      GateWay_SendEmergencyMsg();
    }
```

```
  }
}
```

1.7.10.6. SPO2 Task

```
void SPO2_Algorithm_mainTask(void *pvParameters)
{
  uint32 irBuffer[100];  //infrared LED sensor data
  uint32 redBuffer[100]; //red LED sensor data
  sint32 bufferLength;   //data length
  sint32 spo2;           //SPO2 value
  sint8 validSPO2;       //indicator to show if the SPO2 calculation is valid
  sint32 heartRate;      //heart rate value
  sint8 validHeartRate;  //indicator to show if the heart rate calculation is val
  uint32 NotifiedVal;

  bufferLength = 100; //buffer length of 100 stores 4 seconds of samples running
at 25sps

  //read the first 100 samples, and determine the signal range
  for (uint8 i = 0; i < bufferLength; i++)
  {
    while (MAX30102_available() == FALSE) //do we have new data?
      MAX30102_check();                   //Check the sensor for new data

    redBuffer[i] = MAX30102_getRed();
    irBuffer[i] = MAX30102_getIR();
    MAX30102_nextSample(); //We're finished with this sample so move to next samp
le
  }

  //calculate heart rate and SpO2 after first 100 samples (first 4 seconds of sam
ples)
  maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2
, &validSPO2, &heartRate, &validHeartRate);
  if ((validHeartRate == 1) && (validSPO2 == 1))
  {
    xQueueOverwrite(Spo2Queue, &spo2);
    xQueueOverwrite(HRQueue, &heartRate);
  }

  //Continuously taking samples from MAX30102.  Heart rate and SpO2 are calculate
d every 1 second
```

```
  while (1)
  {
    if ((NotifiedVal & NEW_CARD_PRESENT_NOTIFICATION) == NEW_CARD_PRESENT_NOTIFIC
ATION)
    {
      //dumping the first 25 sets of samples in the memory and shift the last 75
sets of samples to the top
      for (uint8 i = 25; i < 100; i++)
      {
        redBuffer[i - 25] = redBuffer[i];
        irBuffer[i - 25] = irBuffer[i];
      }

      //take 25 sets of samples before calculating the heart rate.
      for (uint8 i = 75; i < 100; i++)
      {
        while (MAX30102_available() == FALSE) //do we have new data?
          MAX30102_check();                   //Check the sensor for new data
        redBuffer[i] = MAX30102_getRed();
        irBuffer[i] = MAX30102_getIR();
      }
      //After gathering 25 new samples recalculate HR and SP02
      maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &
spo2, &validSPO2, &heartRate, &validHeartRate);

      if ((validHeartRate == 1) && (validSPO2 == 1))
      {
        xQueueOverwrite(Spo2Queue, &spo2);
        xQueueOverwrite(HRQueue, &heartRate);
      }
      // Check if no card present
    }
    else
    {
      xTaskNotifyWait(0xffffffff, 0x00, &NotifiedVal, portMAX_DELAY);
    }
  }
  return;
}
```

1.7.11. Monitoring Application

Finally, now that we have implemented what each task should do, we need only to create these tasks, assigns their priorities, stack size, and run the scheduler. This is done in the main function of the application as following:

```
int main(void)
{
  /*  Enable Clock    */
  RCC_clkConfig();

  /*  pins configuration  */
  GPIO_config();

  /* Init MCAL modules */
  I2C_voidPeripheralControl(I2C1, I2C_ENABLE);
  I2C_voidInit(I2C1);
  TIMER_voidInit();
  SPI1_voidInit();
  SPI2_voidInit();
  UART_voidInit();

  /*  Init SWC        */
  Observer_init();
  Display_init();
  Reader_init();
  GateWay_init();

  // CREATE RTOS TASKS
  xTaskCreate(
      Observer_mainTask,        /* Function that implements the task. */
      "OBSERVER",               /* Text name for the task. */
      OBSERVER_TASK_STACK_SIZE, /* Stack size in words, not bytes. */
      (void *)1,                /* Parameter passed into the task. */
      Observer_PRIORITY,        /* Priority at which the task is created. */
      &ObserverHandle);         /* Used to pass out the created task's handle. */
  xTaskCreate(
      SPO2_Algorithm_mainTask, /* Function that implements the task. */
      "SPO2",                  /* Text name for the task. */
      SPO2_TASK_STACK_SIZE,    /* Stack size in words, not bytes. */
      (void *)1,               /* Parameter passed into the task. */
      SPO2_PRIORITY,           /* Priority at which the task is created. */
      &Spo2Handle);            /* Used to pass out the created task's handle. */

  xTaskCreate(
      Display_mainTask,        /* Function that implements the task. */
```

```
        "DISPLAY",                /* Text name for the task. */
        DISPLAY_TASK_STACK_SIZE, /* Stack size in words, not bytes. */
        (void *)1,                /* Parameter passed into the task. */
        Display_PRIORITY,         /* Priority at which the task is created. */
        &DisplayHandle);          /* Used to pass out the created task's handle. */
    xTaskCreate(
        Reader_MainTask,         /* Function that implements the task. */
        "READER",                /* Text name for the task. */
        READER_TASK_STACK_SIZE, /* Stack size in words, not bytes. */
        (void *)1,               /* Parameter passed into the task. */
        Reader_PRIORITY,         /* Priority at which the task is created. */
        &ReaderHandle);          /* Used to pass out the created task's handle. */

    xTaskCreate(
        GateWay_mainTask,         /* Function that implements the task. */
        "GATEWAY",                /* Text name for the task. */
        GATEWAY_TASK_STACK_SIZE, /* Stack size in words, not bytes. */
        (void *)1,                /* Parameter passed into the task. */
        GateWay_PRIORITY,         /* Priority at which the task is created. */
        &GateWayHandle);          /* Used to pass out the created task's handle. */
    xTaskCreate(
        Emergency_mainTask,         /* Function that implements the task. */
        "EMERGENCY",                /* Text name for the task. */
        EMERGENCY_TASK_STACK_SIZE, /* Stack size in words, not bytes. */
        (void *)1,                  /* Parameter passed into the task. */
        Emergency_PRIORITY,         /* Priority at which the task is created. */
        &EmergencyHandle);          /* Used to pass out the created task's handle. *
/

    // Create Queues
    xCardIdQueue = xQueueCreate(ID_READ_LEN, sizeof(loginData_Type));
    xSensorsDataumQueue = xQueueCreate(1, sizeof(ObserverReadingsType));
    Spo2Queue = xQueueCreate(1, sizeof(sint32));
    HRQueue = xQueueCreate(1, sizeof(sint32));

    /* Start the scheduler. */
    vTaskStartScheduler();
    while (1)
    {
    }
    return 0;
} // End main
```

Chapter 2

Cloud Solution

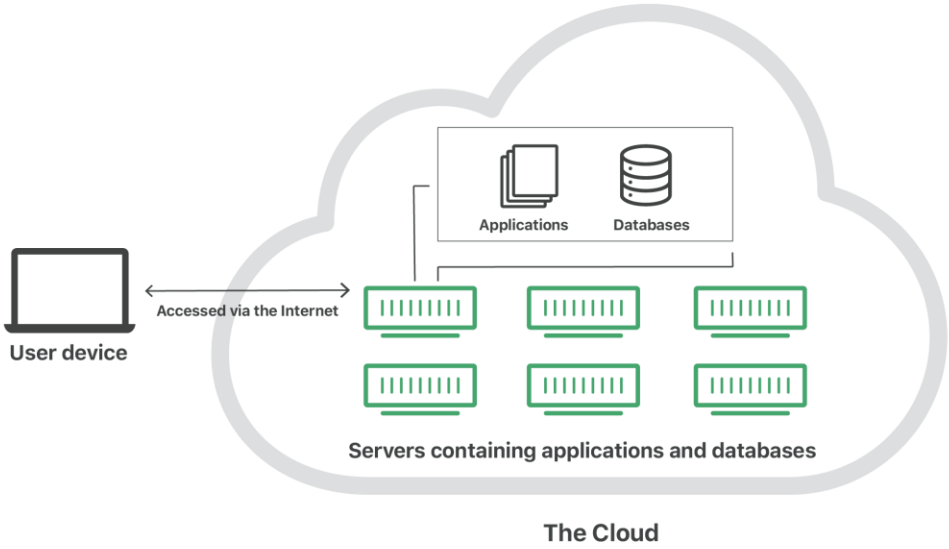
2.1 Introduction

Cloud is a trending topic and widely used technology nowadays. It has become a great business for all large companies as for its large flexibility in resource allocation to the pricing technique. In this chapter we will discuss all about Cloud Solution we made so far and all components we have used to achieve the end goal of the cloud from building the cloud from scratch to deploying the hole infrastructure through one command.

2.2 Cloud

2.2.1 What is the cloud?

"The cloud" refers to servers that are accessed over the Internet, and the software and databases that run on those servers. Cloud servers are in data centers all over the world. By using cloud computing, users and companies do not have to manage physical servers themselves or run software applications on their own machines.



The cloud enables users to access the same files and applications from almost any device, because the computing and storage takes place on servers in a data center, instead of locally on the user device. Therefore, a user can log into their Instagram account on a new phone after their old phone breaks and still find their old account in place, with all their photos, videos, and conversation history. It works the same way

with cloud email providers like Gmail or Microsoft Office 365, and with cloud storage providers like Dropbox or Google Drive.

For businesses, switching to cloud computing removes some IT costs and overhead: for instance, they no longer need to update and maintain their own servers, as the cloud vendor they are using will do that. This especially makes an impact for small businesses that may not have been able to afford their own internal infrastructure but can outsource their infrastructure needs affordably via the cloud. The cloud can also make it easier for companies to operate internationally, because employees and customers can access the same files and applications from any location.

2.2.2 How does cloud computing work?

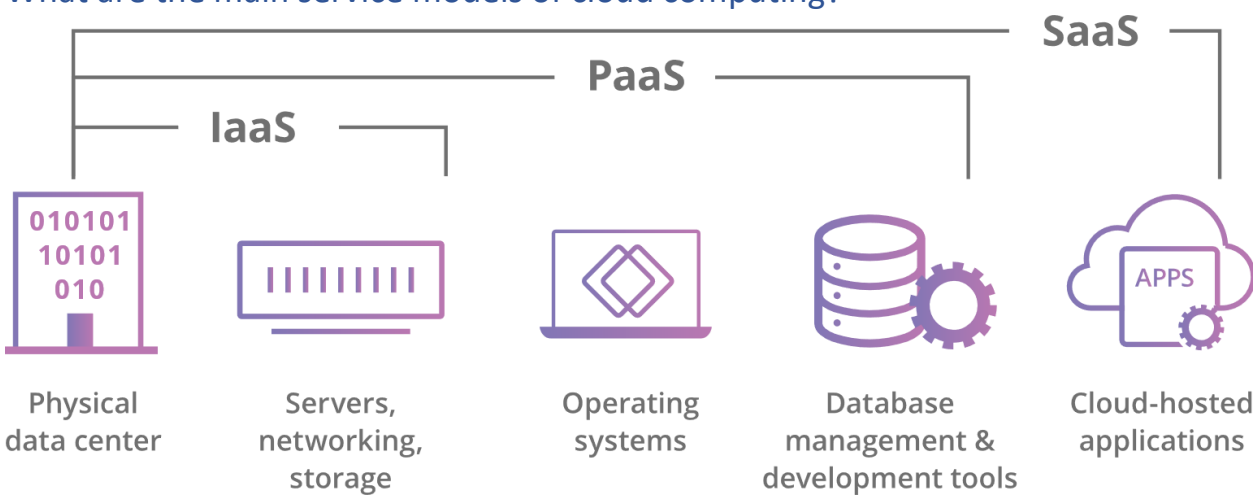
Cloud computing is possible because of a technology called virtualization. Virtualization allows for the creation of a simulated, digital-only "virtual" computer that behaves as if it were a physical computer with its own hardware. The technical term for such a computer is virtual machine. When properly implemented, virtual machines on the same host machine are sandboxed from one another, so they do not interact with each other at all, and the files and applications from one virtual machine are not visible to the other virtual machines even though they are on the same physical machine.

Virtual machines also make more efficient use of the hardware hosting them. By running many virtual machines at once, one server becomes many servers, and a data center becomes a whole host of data centers, able to serve many organizations. Thus, cloud providers can offer the use of their servers to far more customers at once than they would be able to otherwise, and they can do so at a low cost.

Even if individual servers go down, cloud servers in general should be always online and always available. Cloud vendors generally back up their services on multiple machines and across multiple regions.

Users access cloud services either through a browser or through an app, connecting to the cloud over the Internet that is, through many interconnected networks regardless of what device they are using.

2.2.3 What are the main service models of cloud computing?



- Software-as-a-Service (SaaS): Instead of users installing an application on their device, SaaS applications are hosted on cloud servers, and users access them over the Internet. SaaS is like renting a house: the landlord maintains the house, but the tenant mostly gets to use it as if they owned it. Examples of SaaS applications include Salesforce, MailChimp, and Slack.

- Platform-as-a-Service (PaaS): In this model, companies don't pay for hosted applications; instead, they pay for the things they need to build their own applications. PaaS vendors offer everything necessary for building an application, including development tools, infrastructure, and operating systems, over the Internet. PaaS can be compared to renting all the tools and equipment necessary for building a house, instead of renting the house itself. PaaS examples include Heroku and Microsoft Azure.

- Infrastructure-as-a-Service (IaaS): In this model, a company rents the servers and storage they need from a cloud provider. They then use that cloud infrastructure to build their applications. IaaS is like a company leasing a plot of land on which they can build whatever they want – but they need to provide their own building equipment and materials. IaaS providers include DigitalOcean, Google Compute Engine, and OpenStack.

- Function-as-a-Service (FaaS): FaaS, also known as serverless computing, breaks cloud applications down into even smaller components that only run when they're needed. Imagine if it were possible to rent a house one little bit at a time: for instance, the tenant only pays for the dining room at dinner time, the bedroom while they're sleeping, the living room while they're watching TV, and when they aren't using those rooms, they don't have to pay rent on them. FaaS

or serverless applications still run on servers, as do all these models of cloud computing. But they're called "serverless" because they don't run on dedicated machines, and because the companies building the applications don't have to manage any servers. Also, serverless functions scale up, or duplicate, as more people use the application – imagine if the tenant's dining room could expand on demand when more people come over for dinner! Learn more about serverless computing (FaaS).

2.2.4 Cloud Types

In contrast to the models discussed above, which define how services are offered via the cloud, these different cloud deployment types have to do with where the cloud servers are and who manages them.
The most common cloud deployments are:

- Private cloud: A private cloud is a server, data center, or distributed network wholly dedicated to one organization.
- Public cloud: A public cloud is a service run by an external vendor that may include servers in one or multiple data centers. Unlike a private cloud, public clouds are shared by multiple organizations. Using virtual machines, individual servers may be shared by different companies, a situation that is called "multitenancy" because multiple tenants are renting server space within the same server.
- Hybrid cloud: Hybrid cloud deployments combine public and private clouds, and may even include on-premises legacy servers. An organization may use their private cloud for some services and their public cloud for others, or they may use the public cloud as backup for their private cloud.
- Multicloud: Multicloud is a type of cloud deployment that involves using multiple public clouds. In other words, an organization with a multicloud deployment rents virtual servers and services from several external vendors – to continue the analogy used above, this is like leasing several adjacent plots of land from different landlords. Multicloud deployments can also be hybrid cloud, and vice versa.

2.3 Components

For managing to create our solution as cloud platform we have used a set of tools to achieve our goal and going to explain each of them in upcoming sections.

- Terraform: infrastructure as code (IaC) tool.
- Ansible: automation tool.
- Docker: container engine.
- Kubernetes: orchestration tool for containers.
- Rancher: for managing Kubernetes clusters.

2.3.1 Terraform

2.3.1.1 Overview

Terraform is an infrastructure as code (IaC) tool that allows you to build, change, and version infrastructure safely and efficiently. This includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc. Terraform can manage both existing service providers and custom in-house solutions.

2.3.1.2 Key Features

- Infrastructure as Code
  You describe your infrastructure using Terraform's high-level configuration language in human-readable, declarative configuration files. This allows you to create a blueprint that you can version, share, and reuse.
- Execution Plans
  Terraform generates an execution plan describing what it will do and asks for your approval before making any infrastructure changes. This allows you to review changes before Terraform creates, updates, or destroys infrastructure.
- Resource Graph
  Terraform builds a resource graph and creates or modifies non-dependent

resources in parallel. This allows Terraform to build resources as efficiently as possible and gives you greater insight into your infrastructure.

- Change Automation
  Terraform can apply complex changesets to your infrastructure with minimal human interaction. When you update configuration files, Terraform determines what changed and creates incremental execution plans that respect dependencies.

-

2.3.1.3 Use Cases

- Heroku App Setup
  You can use Terraform to codify the setup required for a Heroku application, ensuring that all the required add-ons are available, but it can go even further: configuring DNSimple to set a CNAME, or setting up Cloudflare as a CDN for the app. Best of all, Terraform can do all of this in under 30 seconds without using a web interface.

- Multi-Tier Applications
  A very common pattern is the N-tier architecture. The most common 2-tier architecture is a pool of web servers that use a database tier. Additional tiers get added for API servers, caching servers, routing meshes, etc. This pattern is used because the tiers can be scaled independently and provide a separation of concerns. Terraform is an ideal tool for building and managing these infrastructures. You can group resources in each tier together and Terraform will automatically handle the dependencies between each tier. For example, Terraform will ensure the database tier is available before provisioning the web servers and that the load balancers are connected to the web nodes. You can then use Terraform to easily scale each tier by modifying the count configuration value. Because resource creation and provisioning is codified and automated, elastically scaling with load becomes trivial.

- Self-Service Clusters
  You can use Terraform configuration to codify the knowledge of how to build and scale a service. You can then share these configurations throughout your organization, enabling customer teams to use Terraform to manage their services.

- Software Demos
  Modern software is increasingly networked and distributed. Although tools like Vagrant exist to build virtualized environments for demos, it is still very challenging to demo software on real infrastructure which more closely matches production environments. Software writers can provide a Terraform configuration to create, provision and bootstrap a demo on cloud providers like AWS. This allows end users to easily demo the software on their own infrastructure, and even enables tweaking parameters like cluster size to more rigorously test tools at any scale.

- Disposable Environments
  It is common practice to have both a production and staging or QA environment. These environments are smaller clones of their production counterpart but are used to test new applications before releasing in production. As the production environment grows larger and more complex, it becomes increasingly onerous to maintain an up-to-date staging environment. Using Terraform, the production environment can be codified and then shared with staging, QA or dev. These configurations can be used to rapidly spin up new environments to test in, and then be easily disposed of. Terraform can help tame the difficulty of maintaining parallel environments and makes it practical to elastically create and destroy them.

- Software Defined Networking
  Software Defined Networking (SDN) is becoming increasingly prevalent in the datacenter, as it provides more control to operators and developers and allows the network to better support the applications running on top. Most SDN implementations have a control layer and infrastructure layer. You can use Terraform to codify the configuration for software defined networks. Terraform can then use this configuration to automatically set up and modify settings by interfacing with the control layer. This allows the configuration to be versioned and changes to be automated. For example, you can use Terraform to configure AWS VPC.

- Resource Schedulers
  In large-scale infrastructures, static assignment of applications to machines becomes increasingly challenging. To solve that problem, there are a number of schedulers like Borg, Mesos, YARN, and Kubernetes. These can be used to dynamically schedule Docker containers, Hadoop, Spark, and many other software tools. Terraform is not limited to physical providers like AWS. Resource schedulers can be treated as a provider, enabling Terraform to request resources from them. This allows Terraform to be used in layers: to setup the physical infrastructure running the schedulers as well as provisioning onto the scheduled grid.

- Multi-Cloud Deployment
  It's often attractive to spread infrastructure across multiple clouds to increase fault-tolerance. By using only a single region or cloud provider, fault tolerance is limited by the availability of that provider. Multi-cloud deployment allows for more graceful recovery of the loss of a region or entire provider. Realizing multi-cloud deployments can be very challenging as many existing tools for infrastructure management are cloud-specific. Terraform is cloud-agnostic and allows a single configuration to be used to manage multiple providers, and to even handle cross-cloud dependencies. This simplifies management and orchestration, helping operators build large-scale multi-cloud infrastructures.

2.3.1.4 About the Terraform Language

The main purpose of the Terraform language is declaring resources, which represent infrastructure objects. All other language features exist only to make the definition of resources more flexible and convenient.

A Terraform configuration is a complete document in the Terraform language that tells Terraform how to manage a given collection of infrastructure. A configuration can consist of multiple files and directories.

The syntax of the Terraform language consists of only a few basic elements:

```
resource "aws_vpc" "main" {
  cidr_block = var.base_cidr_block
}

<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}
```

- Blocks are containers for other content and usually represent the configuration of some kind of object, like a resource. Blocks have a block type, can have zero or more labels, and have a body that contains any number of arguments and nested blocks. Most of Terraform's features are controlled by top-level blocks in a configuration file.
- Arguments assign a value to a name. They appear within blocks.
- Expressions represent a value, either literally or by referencing and combining other values. They appear as values for arguments, or within other expressions.

The Terraform language is declarative, describing an intended goal rather than the steps to reach that goal. The ordering of blocks and the files they are organized into are generally not significant; Terraform only considers implicit and explicit relationships between resources when determining an order of operations.

2.3.1.5 Code

```
# Configure the Azure provider
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "= 2.50.0"
    }
  }
}
```

```
provider "azurerm" {
  features {}
}

# Create a resource group if it doesn't exist
resource "azurerm_resource_group" "myterraformgroup" {
    name     = var.resource_group
    location = var.location

    tags = {
        environment = var.env
    }
}
output "rg_name" { value = azurerm_resource_group.myterraformgroup.name }

# Create virtual network
resource "azurerm_virtual_network" "myterraformnetwork" {
    name                = var.virtual_network_name
    address_space       = [var.address_space]
    location            = var.location
    resource_group_name = azurerm_resource_group.myterraformgroup.name

    tags = {
        environment = var.env
    }
}

# Create subnet
resource "azurerm_subnet" "myterraformsubnet" {
    name                 = "${var.virtual_network_name}Subnet1"
    resource_group_name  = azurerm_resource_group.myterraformgroup.name
    virtual_network_name = azurerm_virtual_network.myterraformnetwork.name
    address_prefixes       = [var.subnet_prefix]
}
output "subnet_prefix" { value = var.subnet_prefix }

# Create an SSH key
resource "tls_private_key" "myterraformsshkey" {
  algorithm = "RSA"
  rsa_bits = 4096
}

# Store private key localy
resource "local_file" "myterraformprivatekey" {
    content  = tls_private_key.myterraformsshkey.private_key_pem
```

```
    filename = "../ansible/private_key.pem"
}

# Create Controller VM
# Create public IPs
resource "azurerm_public_ip" "controllervmpublicip" {
    name                         = "${var.controller_vm_name}PublicIP"
    location                     = var.location
    resource_group_name          = azurerm_resource_group.myterraformgroup.name
    allocation_method            = "Dynamic"

    tags = {
        environment = var.env
    }
}

# Create Network Security Group and rule
resource "azurerm_network_security_group" "controllervmsg" {
    name                = "${var.controller_vm_name}SecurityGroup"
    location            = var.location
    resource_group_name = azurerm_resource_group.myterraformgroup.name

    security_rule {
        name                       = "SSH"
        priority                   = 1001
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_range     = "22"
        source_address_prefix      = "*"
        destination_address_prefix = "*"
    }
    security_rule {
        name                       = "http/s"
        priority                   = 1002
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_ranges    = ["80","443"]
        source_address_prefix      = "*"
        destination_address_prefix = "*"
    }
    security_rule {
```

```
        name                       = "openstack"
        priority                   = 1003
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_ranges    = ["5000","2379-2380","9311","8774-
8778","9292","11211","8000","8004","9511","9696","6080"]
        source_address_prefix      = "*"
        destination_address_prefix = "*"
    }
    security_rule {
        name                       = "cockpit"
        priority                   = 1004
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_range     = "9090"
        source_address_prefix      = "*"
        destination_address_prefix = "*"
    }

    tags = {
        environment = var.env
    }
}

# Create network interface
resource "azurerm_network_interface" "controllervmnic" {
    name                      = "${var.controller_vm_name}NIC"
    location                  = var.location
    resource_group_name       = azurerm_resource_group.myterraformgroup.name

    ip_configuration {
        name                          = "${var.controller_vm_name}NicConfiguratio
n"
        subnet_id                     = azurerm_subnet.myterraformsubnet.id
        private_ip_address_allocation = "Static"
        private_ip_address            = var.controller_vm_private_ip
        public_ip_address_id          = azurerm_public_ip.controllervmpublicip.id
    }

    tags = {
        environment = var.env
```

```
    }
}

# Connect the security group to the network interface
resource "azurerm_network_interface_security_group_association" "controllervmass"
 {
    network_interface_id      = azurerm_network_interface.controllervmnic.id
    network_security_group_id = azurerm_network_security_group.controllervmsg.id
}

# Create virtual machine
resource "azurerm_linux_virtual_machine" "controllervm" {
    name                  = var.controller_vm_name
    location              = var.location
    resource_group_name   = azurerm_resource_group.myterraformgroup.name
    network_interface_ids = [azurerm_network_interface.controllervmnic.id]
    size                  = var.controller_vm_size

    os_disk {
        name              = "${var.controller_vm_name}OsDisk"
        caching           = "ReadWrite"
        storage_account_type = "Premium_LRS"
    }

    source_image_reference {
        publisher = "canonical"
        offer     = "0001-com-ubuntu-server-focal"
        sku       = "20_04-lts"
        version   = "latest"
    }

    computer_name  = "controller"
    admin_username = "azureuser"
    disable_password_authentication = true

    admin_ssh_key {
        username       = "azureuser"
        public_key     = tls_private_key.myterraformsshkey.public_key_openssh
    }

    tags = {
        environment = var.env
    }
}
```

```
output "controller_vm_name" { value = azurerm_linux_virtual_machine.controllervm.
name }
output "controller_vm_private_ip" { value = var.controller_vm_private_ip}

# Create Compute1 VM
# Create public IPs
resource "azurerm_public_ip" "compute1vmpublicip" {
    name                        = "${var.compute1_vm_name}PublicIP"
    location                    = var.location
    resource_group_name         = azurerm_resource_group.myterraformgroup.name
    allocation_method           = "Dynamic"

    tags = {
        environment = var.env
    }
}

# Create Network Security Group and rule
resource "azurerm_network_security_group" "compute1vmsg" {
    name                = "${var.compute1_vm_name}SecurityGroup"
    location            = var.location
    resource_group_name = azurerm_resource_group.myterraformgroup.name

    security_rule {
        name                       = "SSH"
        priority                   = 1001
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_range     = "22"
        source_address_prefix      = "*"
        destination_address_prefix = "*"
    }
    security_rule {
        name                       = "cockpit"
        priority                   = 1004
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_range     = "9090"
        source_address_prefix      = "*"
        destination_address_prefix = "*"
    }
```

```
    tags = {
        environment = var.env
    }
}

# Create network interface
resource "azurerm_network_interface" "compute1vmnic" {
    name                      = "${var.compute1_vm_name}NIC"
    location                  = var.location
    resource_group_name       = azurerm_resource_group.myterraformgroup.name

    ip_configuration {
        name                          = "${var.compute1_vm_name}NicConfiguration"
        subnet_id                     = azurerm_subnet.myterraformsubnet.id
        private_ip_address_allocation = "Static"
        private_ip_address            = var.compute1_vm_private_ip
        public_ip_address_id          = azurerm_public_ip.compute1vmpublicip.id
    }

    tags = {
        environment = var.env
    }
}

# Connect the security group to the network interface
resource "azurerm_network_interface_security_group_association" "compute1vmass" {
    network_interface_id      = azurerm_network_interface.compute1vmnic.id
    network_security_group_id = azurerm_network_security_group.compute1vmsg.id
}

# Create virtual machine
resource "azurerm_linux_virtual_machine" "compute1vm" {
    name                  = var.compute1_vm_name
    location              = var.location
    resource_group_name   = azurerm_resource_group.myterraformgroup.name
    network_interface_ids = [azurerm_network_interface.compute1vmnic.id]
    size                  = var.compute1_vm_size

    os_disk {
        name              = "${var.compute1_vm_name}OsDisk"
        caching           = "ReadWrite"
        storage_account_type = "Premium_LRS"
    }
```

```
    source_image_reference {
        publisher = "canonical"
        offer     = "0001-com-ubuntu-server-focal"
        sku       = "20_04-lts"
        version   = "latest"
    }

    computer_name  = "compute1"
    admin_username = "azureuser"
    disable_password_authentication = true

    admin_ssh_key {
        username       = "azureuser"
        public_key     = tls_private_key.myterraformsshkey.public_key_openssh
    }

    tags = {
        environment = var.env
    }
}
output "compute1_vm_name" { value = azurerm_linux_virtual_machine.compute1vm.name
 }
output "compute1_vm_private_ip" { value = var.compute1_vm_private_ip}

# Create Compute2 VM
# Create public IPs
resource "azurerm_public_ip" "compute2vmpublicip" {
    name                         = "${var.compute2_vm_name}PublicIP"
    location                     = var.location
    resource_group_name          = azurerm_resource_group.myterraformgroup.name
    allocation_method            = "Dynamic"

    tags = {
        environment = var.env
    }
}

# Create Network Security Group and rule
resource "azurerm_network_security_group" "compute2vmsg" {
    name                = "${var.compute2_vm_name}SecurityGroup"
    location            = var.location
    resource_group_name = azurerm_resource_group.myterraformgroup.name

    security_rule {
        name                       = "SSH"
```

```
        priority                   = 1001
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_range     = "22"
        source_address_prefix      = "*"
        destination_address_prefix = "*"
    }
    security_rule {
        name                       = "cockpit"
        priority                   = 1004
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_range     = "9090"
        source_address_prefix      = "*"
        destination_address_prefix = "*"
    }

    tags = {
        environment = var.env
    }
}

# Create network interface
resource "azurerm_network_interface" "compute2vmnic" {
    name                      = "${var.compute2_vm_name}NIC"
    location                  = var.location
    resource_group_name       = azurerm_resource_group.myterraformgroup.name

    ip_configuration {
        name                          = "${var.compute2_vm_name}NicConfiguration"
        subnet_id                     = azurerm_subnet.myterraformsubnet.id
        private_ip_address_allocation = "Static"
        private_ip_address            = var.compute2_vm_private_ip
        public_ip_address_id          = azurerm_public_ip.compute2vmpublicip.id
    }

    tags = {
        environment = var.env
    }
}
```

```
# Connect the security group to the network interface
resource "azurerm_network_interface_security_group_association" "compute2vmass" {
    network_interface_id      = azurerm_network_interface.compute2vmnic.id
    network_security_group_id = azurerm_network_security_group.compute2vmsg.id
}

# Create virtual machine
resource "azurerm_linux_virtual_machine" "compute2vm" {
    name                  = var.compute2_vm_name
    location              = var.location
    resource_group_name   = azurerm_resource_group.myterraformgroup.name
    network_interface_ids = [azurerm_network_interface.compute2vmnic.id]
    size                  = var.compute2_vm_size

    os_disk {
        name              = "${var.compute2_vm_name}OsDisk"
        caching           = "ReadWrite"
        storage_account_type = "Premium_LRS"
    }

    source_image_reference {
        publisher = "canonical"
        offer     = "0001-com-ubuntu-server-focal"
        sku       = "20_04-lts"
        version   = "latest"
    }

    computer_name  = "compute2"
    admin_username = "azureuser"
    disable_password_authentication = true

    admin_ssh_key {
        username       = "azureuser"
        public_key     = tls_private_key.myterraformsshkey.public_key_openssh
    }

    tags = {
        environment = var.env
    }
}
output "compute2_vm_name" { value = azurerm_linux_virtual_machine.compute2vm.name
 }
output "compute2_vm_private_ip" { value = var.compute2_vm_private_ip}

# Create Storage VM
```

```
# Create public IPs
resource "azurerm_public_ip" "storagevmpublicip" {
    name                         = "${var.storage_vm_name}PublicIP"
    location                     = var.location
    resource_group_name          = azurerm_resource_group.myterraformgroup.name
    allocation_method            = "Dynamic"

    tags = {
        environment = var.env
    }
}

# Create Network Security Group and rule
resource "azurerm_network_security_group" "storagevmsg" {
    name                = "${var.storage_vm_name}SecurityGroup"
    location            = var.location
    resource_group_name = azurerm_resource_group.myterraformgroup.name

    security_rule {
        name                       = "SSH"
        priority                   = 1001
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_range     = "22"
        source_address_prefix      = "*"
        destination_address_prefix = "*"
    }
    security_rule {
        name                       = "cockpit"
        priority                   = 1004
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_range     = "9090"
        source_address_prefix      = "*"
        destination_address_prefix = "*"
    }

    tags = {
        environment = var.env
    }
}
```

```
# Create network interface
resource "azurerm_network_interface" "storagevmnic" {
    name                      = "${var.storage_vm_name}NIC"
    location                  = var.location
    resource_group_name       = azurerm_resource_group.myterraformgroup.name

    ip_configuration {
        name                          = "${var.storage_vm_name}NicConfiguration"
        subnet_id                     = azurerm_subnet.myterraformsubnet.id
        private_ip_address_allocation = "Static"
        private_ip_address            = var.storage_vm_private_ip
        public_ip_address_id          = azurerm_public_ip.storagevmpublicip.id
    }

    tags = {
        environment = var.env
    }
}

# Connect the security group to the network interface
resource "azurerm_network_interface_security_group_association" "storagevmass" {
    network_interface_id      = azurerm_network_interface.storagevmnic.id
    network_security_group_id = azurerm_network_security_group.storagevmsg.id
}

# Create virtual machine
resource "azurerm_linux_virtual_machine" "storagevm" {
    name                  = var.storage_vm_name
    location              = var.location
    resource_group_name   = azurerm_resource_group.myterraformgroup.name
    network_interface_ids = [azurerm_network_interface.storagevmnic.id]
    size                  = var.storage_vm_size

    os_disk {
        name              = "${var.storage_vm_name}OsDisk"
        caching           = "ReadWrite"
        storage_account_type = "Premium_LRS"
    }

    source_image_reference {
        publisher = "canonical"
        offer     = "0001-com-ubuntu-server-focal"
        sku       = "20_04-lts"
        version   = "latest"
```

```
    }

    computer_name  = "storage"
    admin_username = "azureuser"
    disable_password_authentication = true

    admin_ssh_key {
        username       = "azureuser"
        public_key     = tls_private_key.myterraformsshkey.public_key_openssh
    }

    tags = {
        environment = var.env
    }
}
output "storage_vm_name" { value = azurerm_linux_virtual_machine.storagevm.name }
output "storage_vm_private_ip" { value = var.storage_vm_private_ip}
```

2.3.2 Ansible

2.3.2.1 Overview

Ansible is a software tool that provides simple but powerful automation for cross-platform computer support. It is primarily intended for IT professionals, who use it for application deployment, updates on workstations and servers, cloud provisioning, configuration management, intra-service orchestration, and nearly anything a systems administrator does on a weekly or daily basis. Ansible doesn't depend on agent software and has no additional security infrastructure, so it's easy to deploy.

2.3.2.2 Why Ansible?

Ansible is:

- Simple
    - Human readable automation
    - No special coding skills needed
    - Tasks executed in order
    - Get productive quickly

- Powerful
  - App deployment
  - Configuration management
  - Workflow orchestration
  - Orchestrate the app lifecycle.
- Agentless
  - Agentless architecture
  - Uses OpenSSH and WinRM
  - No agents to exploit or update
  - Predictable, reliable and secure

Ansible is the most popular open source automation tool on GitHub today with more than a quarter million downloads per month. With over 3,550 contributors submitting new modules all the time, rest assured that what you are automating is covered in Ansible already, or will be very soon.

2.3.2.3 How Ansible Works?

Ansible works by connecting to nodes (clients, servers, or whatever you're configuring) on a network, and then sending a small program called an Ansible module to that node. Ansible executes these modules over SSH and removes them when finished. The only requirement for this interaction is that your Ansible control node has login access to the managed nodes. SSH keys are the most common way to provide access, but other forms of authentication are also supported.

2.3.2.4 Use cases

- Provisioning
  - Your apps have to live somewhere. If you’re pxe booting and kickstarting bare-metal servers or vms, or creating virtual or cloud instances from templates, ansible and red hat® ansible® tower help streamline the process.
- Configuration management

  - Centralizing configuration file management and deployment is a common use case for ansible, and it's how many power users are first introduced to the ansible automation platform.

- Application deployment
  - When you define your application with ansible, and manage the deployment with ansible tower, teams are able to effectively manage the entire application lifecycle from development to production.

- Continuous delivery
  - Creating a ci/cd pipeline requires buy-in from numerous teams. You can't do it without a simple automation platform that everyone in your organization can use. Ansible playbooks keep your applications properly deployed (and managed) throughout their entire lifecycle.

- Security automation
  - When you define your security policy in ansible, scanning and remediation of site-wide security policy can be integrated into other automated processes and instead of being an afterthought, it'll be integral in everything that is deployed.

- Orchestration
  - Configurations alone don't define your environment. You need to define how multiple configurations interact and ensure the disparate pieces can be managed as a whole. Out of complexity and chaos, ansible brings order.

2.3.2.5 Integrations

Historically, "infrastructure" typically meant "Linux servers." Ansible's beginnings were forged with automating Linux, but has quickly evolved to many other mediums and domains. Today, no longer just about

automating physical Linux endpoints, Ansible now provides support for automating across private clouds (physical and virtual endpoints) and containerized workloads. So, no matter what your applications require to run on, Ansible has the means to build, provision, manage, and maintain them.

- Private Cloud Automation (including physical and virtual platforms) Ansible is simple and powerful, allowing users to easily manage various physical devices - including the provisioning of bare metal servers, network devices or even storage devices. Using Ansible's Playbook-based automation is easy, and integrates into your current provisioning infrastructure.
    - Red Hat Enterprise Linux Management
    - Red Hat Enterprise Linux
    - Red Hat Satellite
    - Red Hat Smart Management
    - Red Hat Insights
    - Other Linux Operating Systems: CentOS, Fedora, Ubuntu, and others
    - IBM Operating Systems: IBM z/OS, IBM Power Systems, IBM AIX and more
    - Microsoft Windows and Windows Server
    - Linux Management including Hewlett Packard Enterprise, Cisco UCS
    - Infrastructure as a Service: Red Hat OpenStack Platform
- Virtualization Automation
  Moved to virtualized infrastructure? No problem. Ansible supports all the industry-leading virtualization platforms, including:
    - VMware vSphere
    - Red Hat Virtualization (RHV)

- Vagrant

- Container Automation
  Finally, Ansible also provides robust container and native Kubernetes management, expanding to Red Hat OpenShift Container Platform and other container technologies. Ansible supports many of the most popular Kubernetes frameworks and tooling, including:
    - Red Hat OpenShift Platform, OKD
    - Native Kubernetes management and automation
    - Container Developer Tooling: Buildah, Podman, Knative

2.3.2.6 Code

This code used to deploy the hole infrastructure cloud platform and services used.

```
hosts: all
name: install dependenies
gather_facts: no
become: yes
roles:
openstack

hosts: controller
name: install keystone, glance and placement on controller node
gather_facts: no
become: yes
roles:
keystone
glance
placement
horizon
environment:
OS_USERNAME: admin
OS_PASSWORD: password
OS_PROJECT_NAME: admin
OS_USER_DOMAIN_NAME: Default
OS_PROJECT_DOMAIN_NAME: Default
OS_AUTH_URL: http://controller:5000/v3
OS_IDENTITY_API_VERSION: 3

hosts: all
```

```
name: install nova, neutron and cinder
gather_facts: yes
become: yes
roles:
nova
neutron
cinder
environment:
OS_USERNAME: admin
OS_PASSWORD: password
OS_PROJECT_NAME: admin
OS_USER_DOMAIN_NAME: Default
OS_PROJECT_DOMAIN_NAME: Default
OS_AUTH_URL: http://controller:5000/v3
OS_IDENTITY_API_VERSION: 3

hosts: controller
name: install heat, barbican & magnum on controller node
gather_facts: no
become: yes
roles:
# - heat
# - barbican
# - magnum
environment:
OS_USERNAME: admin
OS_PASSWORD: password
OS_PROJECT_NAME: admin
OS_USER_DOMAIN_NAME: Default
OS_PROJECT_DOMAIN_NAME: Default
OS_AUTH_URL: http://controller:5000/v3
OS_IDENTITY_API_VERSION: 3
```

2.3.3 Docker

2.3.3.1 Overview

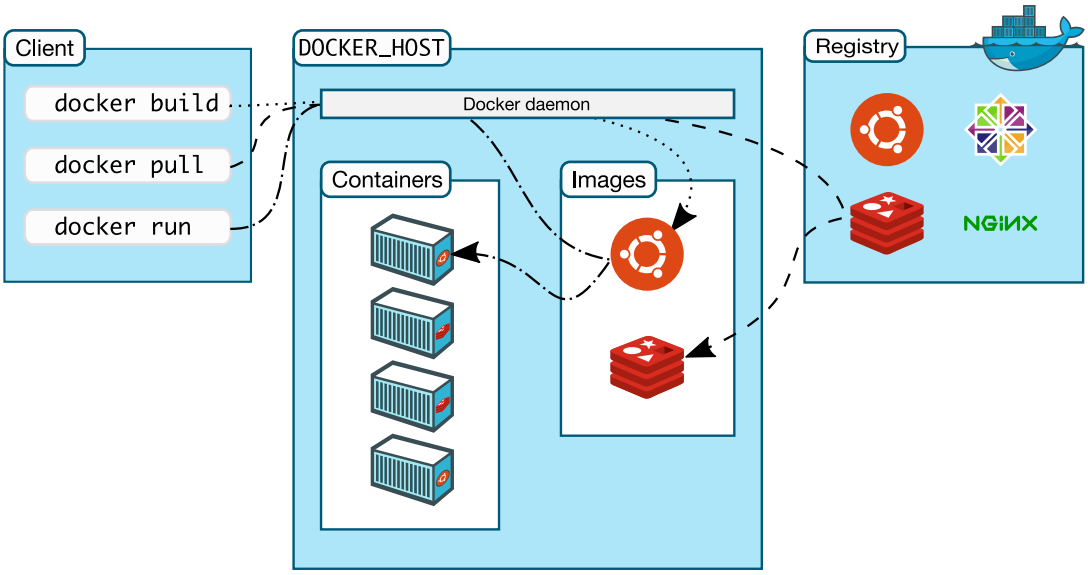
Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

2.3.3.2 What can I use Docker for?

- Fast, consistent delivery of your applications
- Responsive deployment and scaling
- Running more workloads on the same hardware

2.3.3.3 Docker architecture

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



2.3.3.4 The Docker daemon

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

2.3.3.5 The Docker client

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

2.3.3.6 Docker registries

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

2.3.3.7 Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

2.3.3.8 Images

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization.

2.3.3.9 Containers

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

2.3.3.10 The underlying technology

Docker is written in the Go programming language and takes advantage of several features of the Linux kernel to deliver its functionality. Docker uses a technology called namespaces to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container.

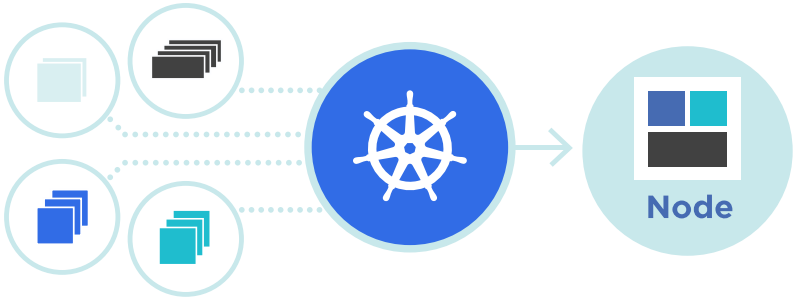
These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

2.3.4 Kubernetes

2.3.4.1 Overview

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

It groups containers that make up an application into logical units for easy management and discovery.



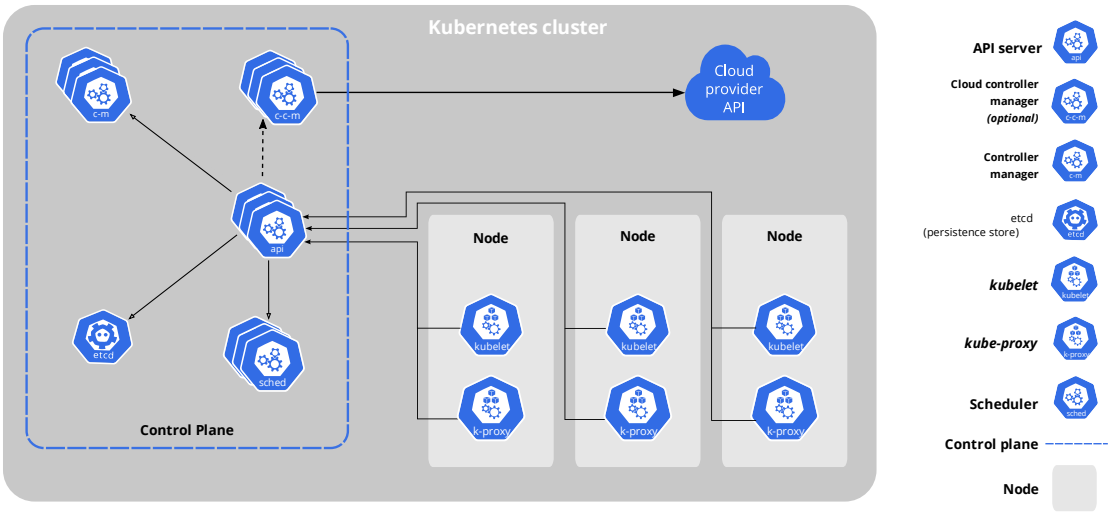
2.3.4.2 Kubernetes Features

- **Automated rollouts and rollbacks**
  - Kubernetes progressively rolls out changes to your application or its configuration, while monitoring application health to ensure it doesn't kill all your instances at the same time. If something goes wrong, Kubernetes will rollback the change for you. Take advantage of a growing ecosystem of deployment solutions.
- **Service discovery and load balancing**
  - No need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives Pods their own IP addresses and a single DNS name for a set of Pods, and can load-balance across them.
- **Storage orchestration**
  - Automatically mount the storage system of your choice, whether from local storage, a public cloud provider such as GCP or AWS, or a network storage system such as NFS, iSCSI, Gluster, Ceph, Cinder, or Flocker.
- **Secret and configuration management**
  - Deploy and update secrets and application configuration without rebuilding your image and without exposing secrets in your stack configuration.
- **Automatic bin packing**
  - Automatically places containers based on their resource requirements and other constraints, while not sacrificing availability. Mix critical and best-effort workloads in order to drive up utilization and save even more resources.
- **Batch execution**
  - In addition to services, Kubernetes can manage your batch and CI workloads, replacing containers that fail, if desired.
- **IPv4/IPv6 dual-stack**
  - Allocation of IPv4 and IPv6 addresses to Pods and Services.

- **Horizontal scaling**
  - Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage.
- **Self-healing**
  - Restarts containers that fail, replaces and reschedules containers when nodes die, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
- **Designed for extensibility**
  - Add features to your Kubernetes cluster without changing upstream source code.

2.3.4.3 Kubernetes Components

When you deploy Kubernetes, you get a cluster. A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node. The worker node(s) host the Pods that are the components of the application workload.
The control plane manages the worker nodes and the Pods in the cluster. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability



2.3.4.4 Control Plane Components

The control plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied).

Control plane components can be run on any machine in the cluster. However, for simplicity, set up scripts typically start all control plane components on the same machine, and do not run user containers on this machine. See Creating Highly Available clusters with kubeadm for an example control plane setup that runs across multiple VMs.

- **kube-apiserver**
    - The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane. The main implementation of a Kubernetes API server is kube-apiserver. kube-apiserver is designed to scale horizontally—that is, it scales by deploying more instances. You can run several instances of kube-apiserver and balance traffic between those instances.

- **Etcd**
    - Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data. If your Kubernetes cluster uses etcd as its backing store, make sure you have a back up plan for those data.
- **kube-scheduler**
    - Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on. Factors taken into account for scheduling decisions include: individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.

- **kube-controller-manager**

- Control Plane component that runs controller processes. Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process. Some types of these controllers are:
    - Node controller: Responsible for noticing and responding when nodes go down.
    - Job controller: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.
    - Endpoints controller: Populates the Endpoints object (that is, joins Services & Pods).
    - Service Account & Token controllers: Create default accounts and API access tokens for new namespaces.

- **cloud-controller-manager**
    - A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster. The cloud-controller-manager only runs controllers that are specific to your cloud provider. If you are running Kubernetes on your own premises, or in a learning environment inside your own PC, the cluster does not have a cloud controller manager. As with the kube-controller-manager, the cloud-controller-manager combines several logically independent control loops into a single binary that you run as a single process. You can scale horizontally (run more than one copy) to improve performance or to help tolerate failures. The following controllers can have cloud provider dependencies:
    - Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding.
    - Route controller: For setting up routes in the underlying cloud infrastructure.

- Service controller: For creating, updating and deleting cloud provider load balancers.

2.3.4.5 Node Components

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

- **Kubelet**
  - An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.
- **kube-proxy**
  - kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept. kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster. kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.
- **Container runtime**
  - The container runtime is the software that is responsible for running containers. Kubernetes supports several container runtimes: Docker, containerd, CRI-O, and any implementation of the Kubernetes CRI (Container Runtime Interface).

2.3.5 OpenStack

2.3.5.1 Overview

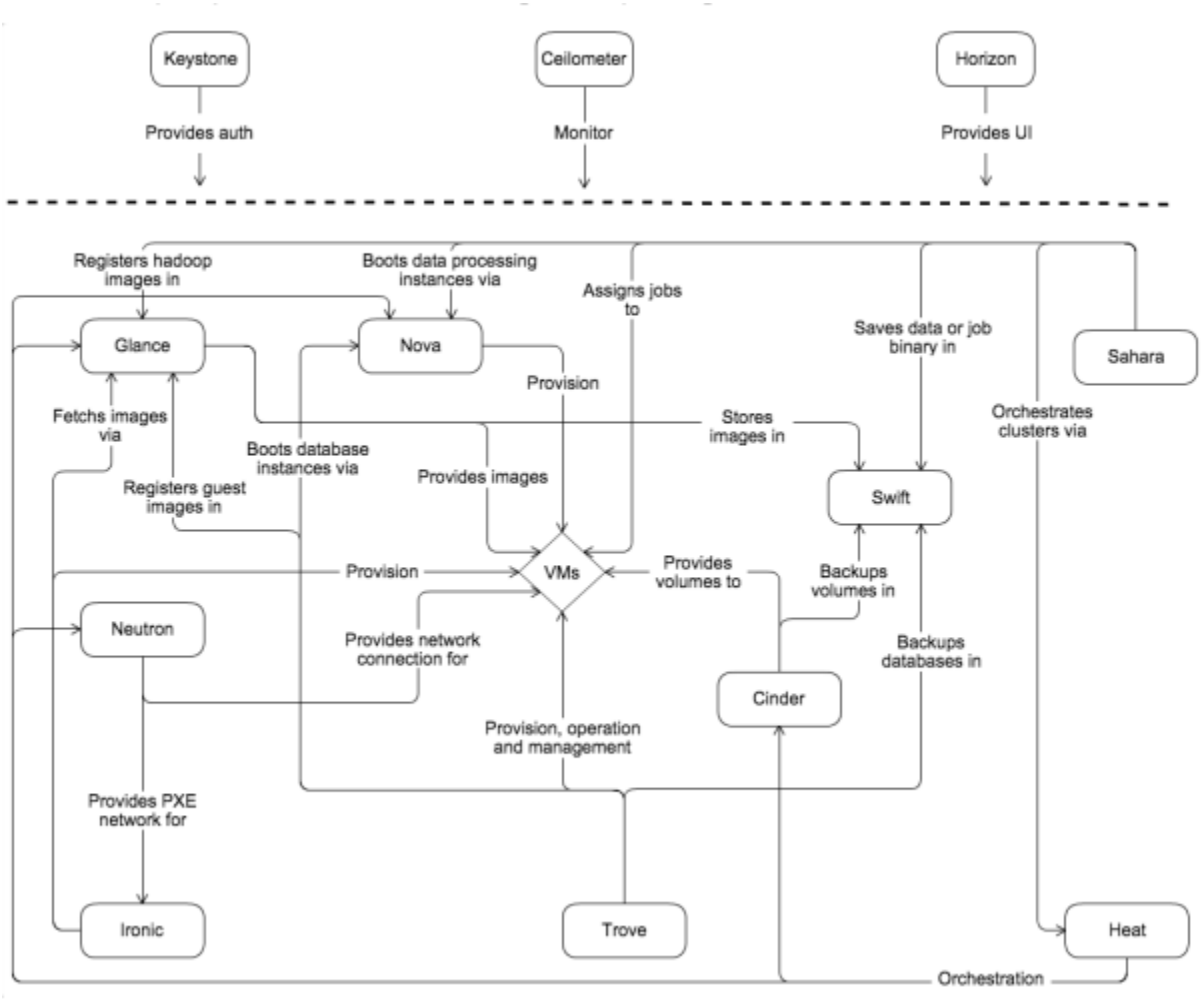
The OpenStack project is an open source cloud computing platform that supports all types of cloud environments. The project aims for simple

implementation, massive scalability, and a rich set of features. Cloud computing experts from around the world contribute to the project. OpenStack provides an Infrastructure-as-a-Service (IaaS) solution through a variety of complementary services. Each service offers an Application Programming Interface (API) that facilitates this integration.

2.3.5.2 The OpenStack architecture

- Conceptual architecture
- Example architecture
  The example architecture requires at least two nodes (hosts) to launch a basic virtual machine or instance. Optional services such as Block Storage and Object Storage require additional nodes. This example architecture differs from a minimal production architecture as follows:
  - Networking agents reside on the controller node instead of one or more dedicated network nodes.
  - Overlay (tunnel) traffic for self-service networks traverses the management network instead of a dedicated network.

The following diagram shows the relationships among the OpenStack services:

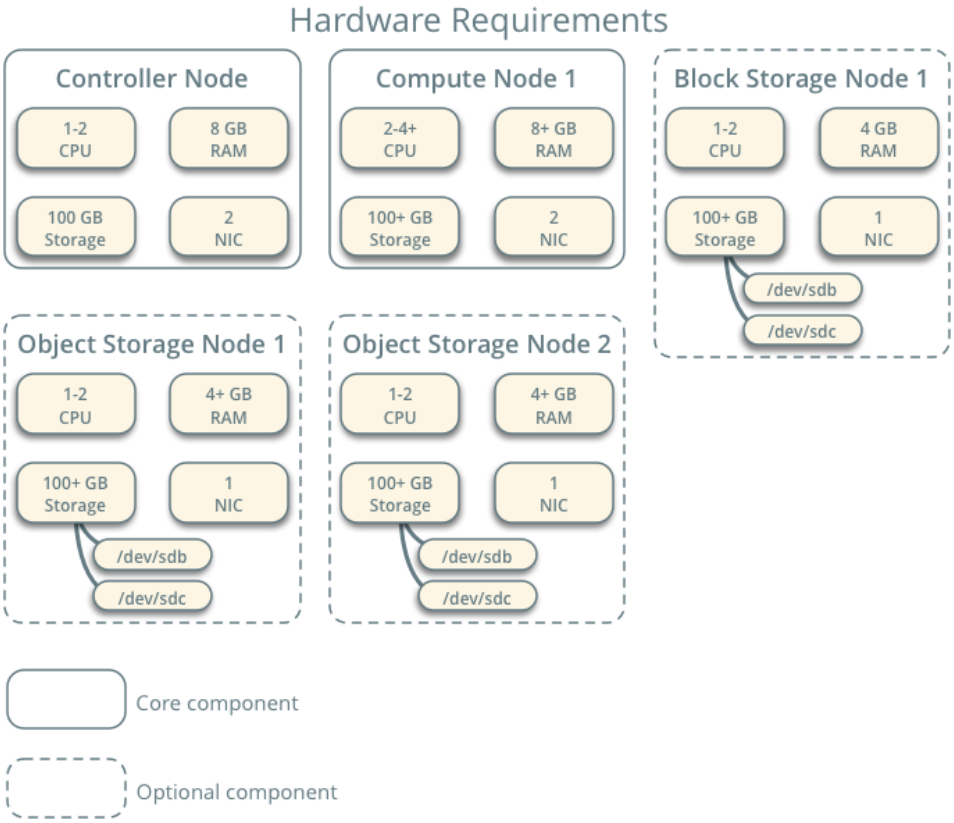


1. Controller

   The controller node runs the Identity service, Image service, Placement service, management portions of Compute, management portion of Networking, various Networking agents, and the Dashboard. It also includes supporting services such as an SQL database, message queue, and NTP. Optionally, the controller node runs portions of the Block Storage, Object Storage, Orchestration, and Telemetry services. The controller node requires a minimum of two network interfaces.

2. Compute

The compute node runs the hypervisor portion of Compute that operates instances. By default, Compute uses the KVM hypervisor. The compute node also runs a Networking service agent that connects instances to virtual networks and provides firewalling services to instances via security groups. You can deploy more than one compute node. Each node requires a minimum of two network interfaces.



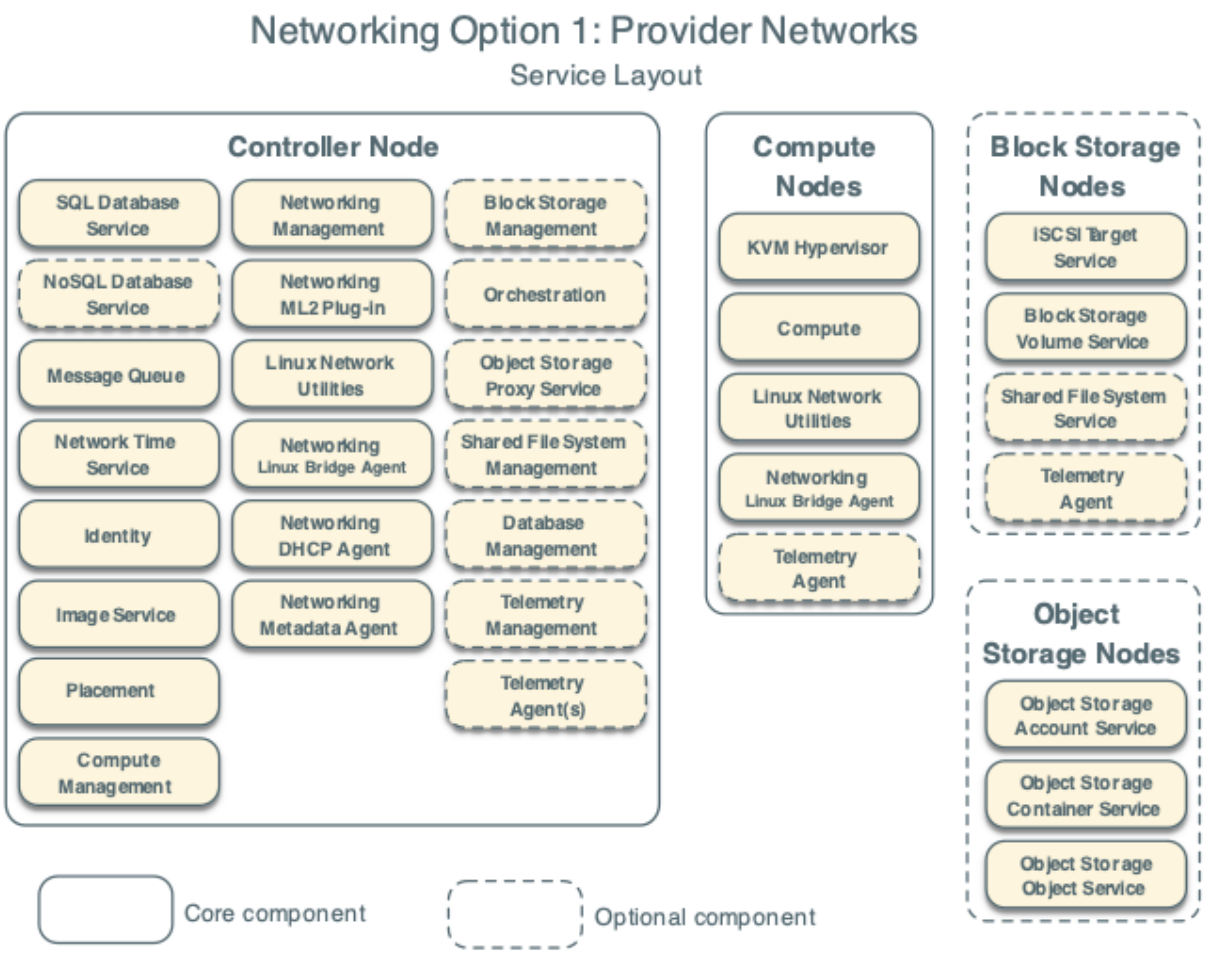
3. Block Storage

The optional Block Storage node contains the disks that the Block Storage and Shared File System services provision for instances. For simplicity, service traffic between compute nodes and this node uses the management network. Production environments should implement a separate storage network to increase performance and security. You can deploy more than one block storage node. Each node requires a minimum of one network interface.

4. Object Storage
   The optional Object Storage node contain the disks that the Object Storage service uses for storing accounts, containers, and objects. For simplicity, service traffic between compute nodes and this node uses the management network. Production environments should implement a separate storage network to increase performance and security. This service requires two nodes. Each node requires a minimum of one network interface. You can deploy more than two object storage nodes.

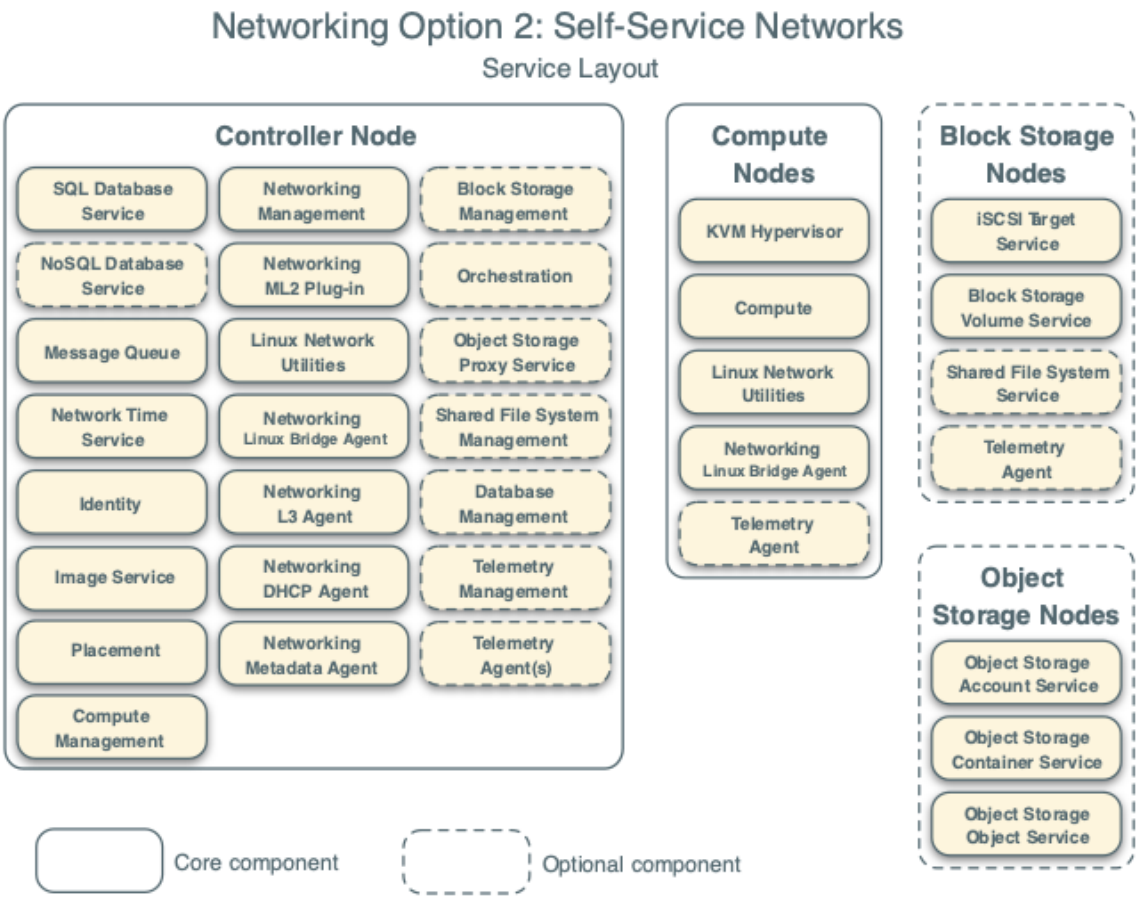
5. Networking
   Choose one of the following virtual networking options.

   a. Networking Option 1: Provider networks
      The provider networks option deploys the OpenStack Networking service in the simplest way possible with primarily layer-2 (bridging/switching) services and VLAN segmentation of networks. Essentially, it bridges virtual



networks to physical networks and relies on physical network infrastructure for layer-3 (routing) services. Additionally, a DHCP service provides IP address information to instances. The OpenStack user requires more information about the underlying network infrastructure to create a virtual network to exactly match the infrastructure.

b. Networking Option 2: Self-service networks
The self-service networks option augments the provider networks option with layer-3 (routing) services that enable self-service networks using overlay segmentation methods such as VXLAN. Essentially, it routes virtual networks to physical networks using NAT. Additionally, this option provides the foundation for advanced services such as LBaaS and FWaaS. The OpenStack user can create virtual networks without the knowledge of underlying infrastructure on the data network. This can also include VLAN networks if the layer-2 plug-in is configured accordingly.



2.3.5.3 Environment

- Security

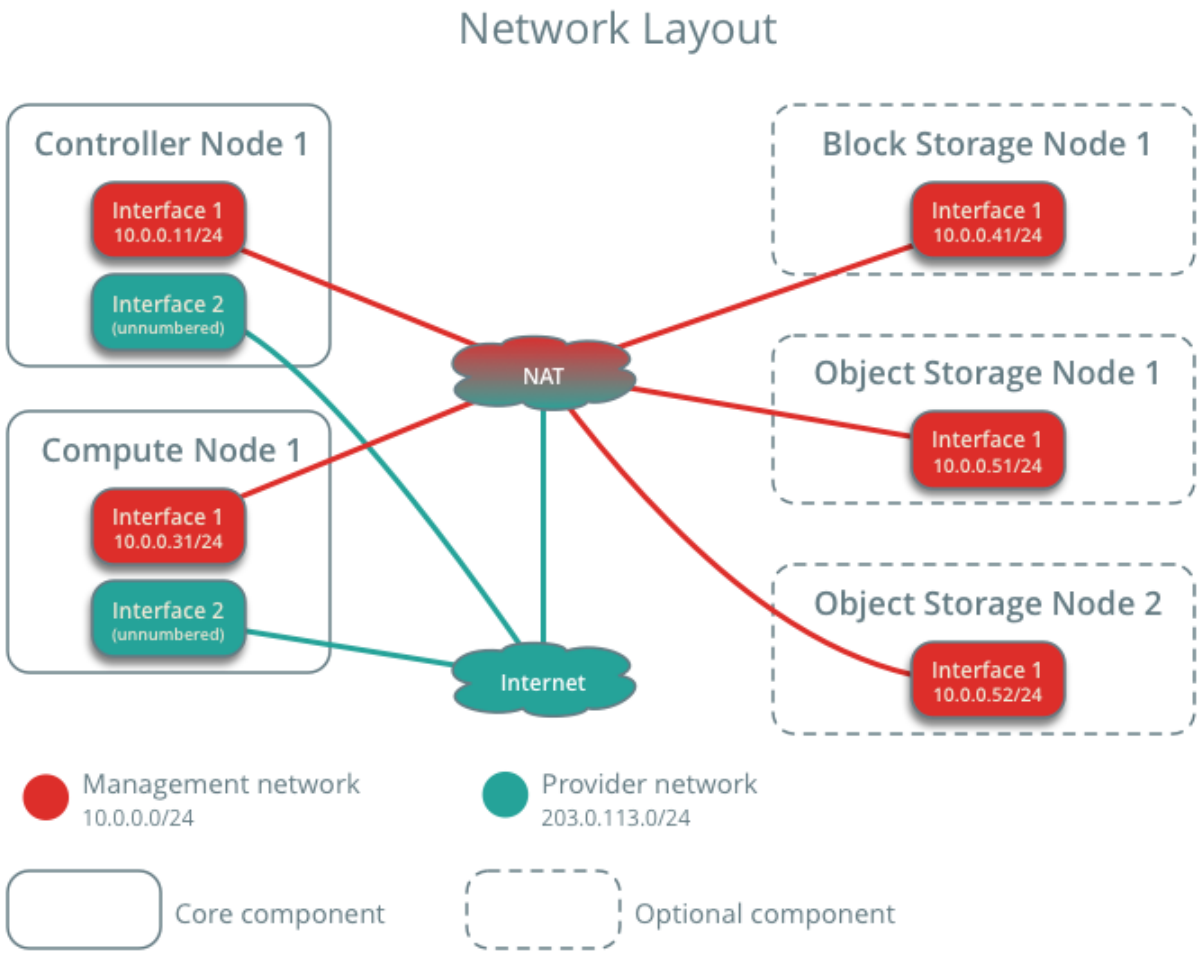
  OpenStack services support various security methods including password, policy, and encryption. Additionally, supporting services including the database server and message broker support password security. OpenStack and supporting services require administrative privileges during installation and operation. In some cases, services perform modifications to the host that can interfere with deployment automation tools such as Ansible, Chef, and Puppet. For example, some OpenStack services add a root wrapper to sudo that can interfere with security policies.

  The Networking service assumes default values for kernel network parameters and modifies firewall rules. To avoid most issues during your initial installation, we recommend using a stock deployment of a supported distribution on your hosts. However, if you choose to automate deployment of your hosts, review the configuration and policies applied to them before proceeding further.

- Host networking

  After installing the operating system on each node for the architecture that you choose to deploy, you must configure the network interfaces. We recommend that you disable any automated network management tools and manually edit the appropriate configuration files for your distribution. All nodes require Internet access for administrative purposes such as package installation, security updates, DNS, and NTP. In most cases, nodes should obtain Internet access through the management network interface. To highlight the importance of network separation, the example architectures use private address space for the management network and assume that the physical network infrastructure provides Internet access via NAT or other methods. The example architectures use routable IP address space for the provider (external) network and assume that the physical network infrastructure provides direct Internet access. In the provider networks architecture, all instances attach directly to the provider network. In the selfservice (private) networks architecture, instances can attach to a self-service or provider network. Selfservice networks can reside entirely within

OpenStack or provide some level of external network access using NAT through the provider network.



- Network Time Protocol (NTP)
  To properly synchronize services among nodes, you can install Chrony, an implementation of NTP. We recommend that you configure the controller node to reference more accurate (lower stratum) servers and other nodes to reference the controller node.

- OpenStack packages
  Distributions release OpenStack packages as part of the distribution or using other methods because of differing release schedules. Perform these procedures on all nodes.

2.3.5.4 OpenStack Services

- **Block Storage service (Cinder)**
  The Block Storage service (cinder) provides block storage devices to guest instances. The method in which the storage is provisioned and consumed is determined by the Block Storage driver, or drivers in the case of a multi-backend configuration. There are a variety of drivers that are available: NAS/SAN, NFS, iSCSI, Ceph, and more. The Block Storage API and scheduler services typically run on the controller nodes. Depending upon the drivers used, the volume service can run on controller nodes, compute nodes, or standalone storage nodes. The OpenStack Block Storage service (Cinder) adds persistent storage to a virtual machine. Block Storage provides an infrastructure for managing volumes, and interacts with OpenStack Compute to provide volumes for instances. The service also enables management of volume snapshots, and volume types.
  The Block Storage service consists of the following components:

  - **cinder-api**
    Accepts API requests, and routes them to the cinder-volume for action.

  - **cinder-volume**
    Interacts directly with the Block Storage service, and processes such as the cinder-scheduler. It also interacts with these processes through a message queue. The cinder-volume service responds to read and write requests sent to the Block Storage service to maintain state. It can interact with a variety of storage providers through a driver architecture.

  - **cinder-scheduler daemon**
    Selects the optimal storage provider node on which to create the volume. A similar component to the nova-scheduler.

  - **cinder-backup daemon**
    The cinder-backup service provides backing up volumes of any type to a backup storage provider. Like the cinder-volume service, it can interact with a variety of storage providers through a driver architecture.

- **Messaging queue**
  Routes information between the Block Storage processes.

- **Compute service (Nova)**

  Use OpenStack Compute to host and manage cloud computing systems. OpenStack Compute is a major part of an Infrastructure-as-a-Service (IaaS) system. The main modules are implemented in Python. OpenStack Compute interacts with OpenStack Identity for authentication, OpenStack Placement for resource inventory tracking and selection, OpenStack Image service for disk and server images, and OpenStack Dashboard for the user and administrative interface. Image access is limited by projects, and by users; quotas are limited per project (the number of instances, for example). OpenStack Compute can scale horizontally on standard hardware, and download images to launch instances.
  OpenStack Compute consists of the following areas and their components:

  - **nova-api service**
    Accepts and responds to end user compute API calls. The service supports the OpenStack Compute API. It enforces some policies and initiates most orchestration activities, such as running an instance.

  - **nova-api-metadata service**
    Accepts metadata requests from instances. For more information, refer to Metadata service.

  - **nova-compute service**
    A worker daemon that creates and terminates virtual machine instances through hypervisor APIs. For example:

    - libvirt for KVM or QEMU

    - VMwareAPI for VMware

Processing is fairly complex. Basically, the daemon accepts actions from the queue and performs a series of system commands such as launching a KVM instance and updating its state in the database.

- **nova-scheduler service**

  Takes a virtual machine instance request from the queue and determines on which compute server host it runs.

- **nova-conductor module**

  Mediates interactions between the nova-compute service and the database. It eliminates direct accesses to the cloud database made by the nova-compute service. The nova-conductor module scales horizontally. However, do not deploy it on nodes where the nova-compute service runs. For more information, see the conductor section in the Configuration Options.

- **nova-novncproxy daemon**

  Provides a proxy for accessing running instances through a VNC connection. Supports browser-based novnc clients.

- **nova-spicehtml5proxy daemon**

  Provides a proxy for accessing running instances through a SPICE connection. Supports browser-based HTML5 client.

- **The queue**

  A central hub for passing messages between daemons. Usually implemented with RabbitMQ but other options are available.

- **SQL database**

  Stores most build-time and run-time states for a cloud infrastructure, including:

  - Available instance types
  - Instances in use
  - Available networks
  - Projects

Theoretically, OpenStack Compute can support any database that SQLAlchemy supports. Common databases are SQLite3 for test and development work, MySQL, MariaDB, and PostgreSQL.

- **Dashboard (Horizon)**
  OpenStack Horizon is a web-based graphical interface that cloud administrators and users can access to manage OpenStack compute, storage and networking services. The Horizon interface works with Openstack Nova, a compute service; OpenStack Swift, an object storage system; OpenStack Keystone, an identify management service; and many others. It accesses these services via OpenStack application programming interfaces (APIs). OpenStack Horizon offers three versions of management dashboards: a User Dashboard, a System Dashboard and a Settings Dashboard. Cloud admins are able to customize visual elements of the Horizon interface, including the navigation bar, tables, alerts and other elements. Developers can also extend any existing dashboard to include more functionality by building an application that integrates with the dashboard.
  The only core service required by the dashboard is the Identity service. You can use the dashboard in combination with other services, such as Image service, Compute, and Networking. You can also use the dashboard in environments with stand-alone services such as Object Storage.

- **Identity service (Keystone)**
  The OpenStack system consists of several key services that are separately installed. These services work together depending on your cloud needs and include the Compute, Identity, Networking, Image, Block Storage, Object Storage, Telemetry, Orchestration, and Database services. You can install any of these projects separately and configure them stand-alone or as connected entities. This section describes how to install and configure the OpenStack Identity service, code-named keystone, on the controller node. For scalability purposes, this configuration deploys Fernet tokens and the Apache HTTP server to handle requests.

The OpenStack Identity service provides a single point of integration for managing authentication, authorization, and a catalog of services. The Identity service is typically the first service a user interacts with. Once authenticated, an end user can use their identity to access other OpenStack services. Likewise, other OpenStack services leverage the Identity service to ensure users are who they say they are and discover where other services are within the deployment. The Identity service can also integrate with some external user management systems (such as LDAP).

Users and services can locate other services by using the service catalog, which is managed by the Identity service. As the name implies, a service catalog is a collection of available services in an OpenStack deployment. Each service can have one or many endpoints and each endpoint can be one of three types: admin, internal, or public. In a production environment, different endpoint types might reside on separate networks exposed to different types of users for security reasons. For instance, the public API network might be visible from the Internet so customers can manage their clouds. The admin API network might be restricted to operators within the organization that manages cloud infrastructure. The internal API network might be restricted to the hosts that contain OpenStack services. Also, OpenStack supports multiple regions for scalability. For simplicity, this guide uses the management network for all endpoint types and the default RegionOne region. Together, regions, services, and endpoints created within the Identity service comprise the service catalog for a deployment. Each OpenStack service in your deployment needs a service entry with corresponding endpoints stored in the Identity service. This can all be done after the Identity service has been installed and configured. The Identity service contains these components:

- **Server**
  A centralized server provides authentication and authorization services using a RESTful interface.

- **Drivers**
  Drivers or a service back end are integrated to the centralized server. They are used for accessing identity information in repositories external to OpenStack, and may already exist in the

infrastructure where OpenStack is deployed (for example, SQL databases or LDAP servers).

- **Middleware**
  modules run in the address space of the OpenStack component that is using the Identity service. These modules intercept service requests, extract user credentials, and send them to the centralized server for authorization. The integration between the middleware modules and OpenStack components uses the Python Web Server Gateway Interface.

- **Image service (Glance)**
  The Image service (glance) enables users to discover, register, and retrieve virtual machine images. It offers a REST API that enables you to query virtual machine image metadata and retrieve an actual image. You can store virtual machine images made available through the Image service in a variety of locations, from simple file systems to object-storage systems like OpenStack Object Storage.
  The OpenStack Image service is central to Infrastructure-as-a-Service (IaaS). It accepts API requests for disk or server images, and metadata definitions from end users or OpenStack Compute components. It also supports the storage of disk or server images on various repository types, including OpenStack Object Storage.
  A number of periodic processes run on the OpenStack Image service to support caching. Replication services ensure consistency and availability through the cluster. Other periodic processes include auditors, updaters, and reapers.
  The OpenStack Image service includes the following components:

  - **glance-api**
    Accepts Image API calls for image discovery, retrieval, and storage.

  - **Database**
    Stores image metadata and you can choose your database depending on your preference. Most deployments use MySQL or SQLite.

- **Storage repository for image files**
  Various repository types are supported including normal file systems (or any filesystem mounted on the glance-api controller node), Object Storage, RADOS block devices, VMware datastore, and HTTP. Note that some repositories will only support read-only usage.
- **Metadata definition service**
  A common API for vendors, admins, services, and users to meaningfully define their own custom metadata. This metadata can be used on different types of resources like images, artifacts, volumes, flavors, and aggregates. A definition includes the new property's key, description, constraints, and the resource types which it can be associated with.

- **Key Manager service (Barbican)**

  The Key Manager service provides secure storage, provisioning and management of secrets, such as passwords, encryption keys, etc.

  The Key Manager service consists of the following components:

  - **barbican-api service**
    Provides an OpenStack-native RESTful API that supports provisioning and managing Barbican secrets.
  - **barbican-worker service**
    Provides an Openstack RPC interface that interacts with barbican-api and reads from the barbican message queue. Supports the fulfillment of Barbican orders.
  - **barbican-keystone-listener service**
    Listens to messages from the Keystone notification service. Used to manage the representation of Keystone projects in the Barbican database when projects are deleted.

- **Networking service (Neutron)**

  OpenStack Networking (neutron) allows you to create and attach interface devices managed by other OpenStack services to networks. Plug-ins can be implemented to accommodate different networking equipment and software, providing flexibility to OpenStack architecture and deployment.

  It includes the following components:

  neutron-server
  Accepts and routes API requests to the appropriate OpenStack Networking plug-in for action.

  OpenStack Networking plug-ins and agents
  Plug and unplug ports, create networks or subnets, and provide IP addressing. These plug-ins and agents differ depending on the vendor and technologies used in the particular cloud. OpenStack Networking ships with plug-ins and agents for Cisco virtual and physical switches, NEC OpenFlow products, Open vSwitch, Linux bridging, and the VMware NSX product.
  The common agents are L3 (layer 3), DHCP (dynamic host IP addressing), and a plug-in agent.

  Messaging queue
  Used by most OpenStack Networking installations to route information between the neutron-server and various agents. Also acts as a database to store networking state for particular plug-ins.

  OpenStack Networking mainly interacts with OpenStack Compute to provide networks and connectivity for its instances.

Networking (neutron) concepts

OpenStack Networking (neutron) manages all networking facets for the Virtual Networking Infrastructure (VNI) and the access layer aspects of the Physical Networking Infrastructure (PNI) in your OpenStack environment. OpenStack Networking enables projects to create advanced virtual network topologies which may include services such as a firewall, and a virtual private network (VPN).

Networking provides networks, subnets, and routers as object abstractions. Each abstraction has functionality that mimics its physical counterpart: networks contain subnets, and routers route traffic between different subnets and networks.

Any given Networking set up has at least one external network. Unlike the other networks, the external network is not merely a virtually defined network. Instead, it represents a view into a slice of the physical, external network accessible outside the OpenStack installation. IP addresses on the external network are accessible by anybody physically on the outside network.

In addition to external networks, any Networking set up has one or more internal networks. These software-defined networks connect directly to the VMs. Only the VMs on any given internal network, or those on subnets connected through interfaces to a similar router, can access VMs connected to that network directly.

For the outside network to access VMs, and vice versa, routers between the networks are needed. Each router has one gateway that is connected to an external network and one or more interfaces connected to internal networks. Like a physical router, subnets can access machines on other subnets that are connected to the same router, and machines can access the outside network through the gateway for the router.

Additionally, you can allocate IP addresses on external networks to ports on the internal network. Whenever something is connected to a subnet, that connection is called a port. You can associate external network IP addresses with ports to VMs. This way, entities on the outside network can access VMs.

Networking also supports security groups. Security groups enable administrators to define firewall rules in groups. A VM can belong to one or more security groups, and Networking applies the rules in those security groups to block or unblock ports, port ranges, or traffic types for that VM.

Each plug-in that Networking uses has its own concepts. While not vital to operating the VNI and OpenStack environment, understanding these concepts can help you set up Networking. All Networking installations use a core plug-in and a security group plug-in (or just the No-Op security group plug-in).

- **Orchestration service (Heat)**

  The Orchestration service provides a template-based orchestration for describing a cloud application by running OpenStack API calls to generate running cloud applications. The software integrates other core components of OpenStack into a one-file template system. The templates allow you to create most OpenStack resource types such as instances, floating IPs, volumes, security groups, and users. It also provides advanced functionality such as instance high availability, instance auto-scaling, and nested stacks.
  This enables OpenStack core projects to receive a larger user base. The service allows deployers to integrate with the Orchestration service directly or through custom plug-ins.
  The Orchestration service consists of the following components:

  - **heat command-line client**
    A CLI that communicates with the **heat-api** to run AWS CloudFormation APIs. End developers can directly use the Orchestration REST API.
  - **heat-api component**
    An OpenStack-native REST API that processes API requests by sending them to the **heat-engine** over Remote Procedure Call (RPC).
  - **heat-api-cfn component**
    An AWS Query API that is compatible with AWS CloudFormation. It processes API requests by sending them to the **heat-engine** over RPC.
  - **heat-engine**
    Orchestrates the launching of templates and provides events back to the API consumer.

- **Placement service (Placement)**
  - **Placement Usage**
  - **Tracking Resources**

    The placement service enables other projects to track their own resources. Those projects can register/delete their own resources to/from placement via the placement HTTP API.

The placement service originated in the Nova project. As a result, much of the functionality in placement was driven by nova's requirements. However, that functionality was designed to be sufficiently generic to be used by any service that needs to manage the selection and consumption of resources.

- **How Nova Uses Placement**

  Two processes, nova-compute and nova-scheduler, host most of nova's interaction with placement.
  The nova resource tracker in nova-compute is responsible for creating the resource provider record corresponding to the compute host on which the resource tracker runs, setting the inventory that describes the quantitative resources that are available for workloads to consume (e.g., VCPU), and setting the traits that describe qualitative aspects of the resources (e.g., STORAGE_DISK_SSD).
  If other projects – for example, Neutron or Cyborg – wish to manage resources on a compute host, they should create resource providers as children of the compute host provider and register their own managed resources as inventory on those child providers. For more information, see the Modeling with Provider Trees.
  The nova-scheduler is responsible for selecting a set of suitable destination hosts for a workload. It begins by formulating a request to placement for a list of allocation candidates. That request expresses quantitative and qualitative requirements, membership in aggregates, and in more complex cases, the topology of related resources. That list is reduced and ordered by filters and weighers within the scheduler process. An allocation is made against a resource provider representing a destination, consuming a portion of the inventory set by the resource tracker.

- **REST API**

  The placement API service provides a well-documented, JSON-based HTTP API and data model. It is designed to be easy to use from whatever HTTP client is suitable. There is a plugin to the openstack client command line tool called osc-placement which is useful for occasional inspection and manipulation of the resources in the placement service.

- **Microversions**
  The placement API uses microversions for making incremental changes to the API which client requests must opt into. It is especially important to keep in mind that nova-compute is a client of the placement REST API and based on how Nova supports rolling upgrades the nova-compute service could be Newton level code making requests to an Ocata placement API, and vice-versa, an Ocata compute service in a cells v2 cell could be making requests to a Newton placement API.

2.3.5.5 Deployment

We used Ansible playbooks to automate the deployment of OpenStack and its services.
Here is samples of the code the main code and more instruction for deployment is on GitHub link in the references section.

Code for deploying Cinder:

```
- name: Install and configure controller node

  include: controller-install.yml
  when: "'controller' in {{ group_names }}"

- name: Install and configure a storage node
  include: storage-install.yml
  when: "'storage' in {{ group_names }}"

- name:  Install nfs client
  apt:
    name: nfs-common
    state: latest
- name: Verify Cinder operation
  shell: openstack volume service list
  register: cindereg

- name: print previous command output
  debug:
    var: cindereg.stdout_lines
```

Code for deploying Nova:

```
- name: Install and configure controller node
  include: controller-install.yml
  when: "'controller' in {{ group_names }}"

- name: Install and configure a compute node
  include: compute-install.yml
  when: "'compute' in {{ group_names }}"

- name: Discover compute hosts
  shell: su -s /bin/sh -c "nova-manage cell_v2 discover_hosts --
verbose" nova
  when: "'controller' in {{ group_names }}"
  register: discreg

- name: print previous command output
  when: "'controller' in {{ group_names }}"
  debug:
    var: discreg.stdout_lines
```

Code for deploying Horizon:

```
- name: install horizon
  apt:
    name: openstack-dashboard
    state: latest

- name: Edit the /etc/openstack-dashboard/local_settings.py file
  copy:
    src: local_settings.py
    dest: /etc/openstack-dashboard/local_settings.py
    owner: horizon
    group: horizon

- name: Reload the apache2 service
  systemd:
    name: apache2
    state: reloaded
```

Code for deploying Keystone:

```
- name: Create the keystone database
  shell: mysql -e "CREATE DATABASE keystone;"

# TODO update keystone password here
```

```
- name: Grant proper access to the keystone database
  shell: |
    mysql -
e "CREATE USER 'keystone'@'localhost' IDENTIFIED BY 'password';"
    mysql -
e "GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost';"
    mysql -e "CREATE USER 'keystone'@'%' IDENTIFIED BY 'password';"
    mysql -e "GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%';"
#   https://ma.ttias.be/mysql-8-removes-shorthand-creating-user-
permissions/

- name:  install keystone
  apt:
    name: keystone
    state: latest

# TODO update keystone password here
- name: Edit the /etc/keystone/keystone.conf file
  ini_file:
    path: /etc/keystone/keystone.conf
    owner: keystone
    group: keystone
    section: database
    option: connection
    value: mysql+pymysql://keystone:password@localhost/keystone

- name: Populate the Identity service database
  shell: su -s /bin/sh -c "keystone-manage db_sync" keystone

- name: Initialize Fernet key repositories
  shell: |
    keystone-manage fernet_setup --keystone-user keystone --keystone-
group keystone
    keystone-manage credential_setup --keystone-user keystone --keystone-
group keystone

# TODO change admin password
- name: Bootstrap the Identity service
  shell: |
    keystone-manage bootstrap --bootstrap-password password \
      --bootstrap-admin-url http://controller:5000/v3/ \
      --bootstrap-internal-url http://controller:5000/v3/ \
      --bootstrap-public-url http://controller:5000/v3/ \
      --bootstrap-region-id RegionOne
```

```
- name: Configure the Apache HTTP server
  shell: echo "ServerName controller" >> /etc/apache2/apache2.conf

- name: Restart service apache2
  service:
    name: apache2
    state: restarted

- name: create example domain
  shell: openstack domain create --description "An Example Domain" example

- name: Create the service project that contains a unique user for each se
rvice that you add to your environment
  shell: openstack project create --domain default --
description "Service Project" service
```

Code for deploying Glance:

```
- name: Create the glance database
  shell: mysql -e "CREATE DATABASE glance;"

# TODO update glance password here
- name: Grant proper access to the glance database
  shell: |
    mysql -e "CREATE USER 'glance'@'localhost' IDENTIFIED BY 'password';"
    mysql -e "GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'localhost';"
    mysql -e "CREATE USER 'glance'@'%' IDENTIFIED BY 'password';"
    mysql -e "GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%';"
#   https://ma.ttias.be/mysql-8-removes-shorthand-creating-user-
permissions/

# TODO update glance password here
- name: Create the glance user
  shell: openstack user create --domain default --password password glance

- name: Add the admin role to the glance user and service project
  shell: openstack role add --project service --user glance admin

- name: Create the glance service entity
  shell: openstack service create --name glance --
description "OpenStack Image" image

- name: Create the Image service API endpoints
  shell: |
```

```
    openstack endpoint create --
region RegionOne image public http://controller:9292
    openstack endpoint create --
region RegionOne image internal http://controller:9292
    openstack endpoint create --
region RegionOne image admin http://controller:9292

- name:  install glance
  apt:
    name: glance
    state: latest

# TODO update glance password here
- name: Edit the /etc/glance/glance-api.conf file
  ini_file:
    path: /etc/glance/glance-api.conf
    owner: glance
    group: glance
    section: "{{ item.section }}"
    option: "{{ item.option }}"
    value: "{{ item.value }}"
  loop:
    - { section: "database", option: "connection", value: 'mysql+pymysql:/
/glance:password@localhost/glance' }
    - { section: "keystone_authtoken", option: "www_authenticate_uri", val
ue: 'http://controller:5000' }
    - { section: "keystone_authtoken", option: "auth_url", value: 'http://
controller:5000' }
    - { section: "keystone_authtoken", option: "memcached_servers", value:
 'controller:11211' }
    - { section: "keystone_authtoken", option: "auth_type", value: 'passwo
rd' }
    - { section: "keystone_authtoken", option: "project_domain_name", valu
e: 'Default' }
    - { section: "keystone_authtoken", option: "user_domain_name", value:
'Default' }
    - { section: "keystone_authtoken", option: "project_name", value: 'ser
vice' }
    - { section: "keystone_authtoken", option: "username", value: 'glance'
 }
    - { section: "keystone_authtoken", option: "password", value: 'passwor
d' }
    - { section: "paste_deploy", option: "flavor", value: 'keystone' }
    - { section: "glance_store", option: "stores", value: 'file,http' }
    - { section: "glance_store", option: "default_store", value: 'file' }
```

```
    - { section: "glance_store", option: "filesystem_store_datadir", value
: '/var/lib/glance/images/' }

- name: Populate the Image service database
  shell: su -s /bin/sh -c "glance-manage db_sync" glance

- name: Restart service glance-api
  systemd:
    enabled: true
    name: glance-api
    state: restarted

- name: add cirros image
  shell: |
    wget http://download.cirros-cloud.net/0.4.0/cirros-0.4.0-x86_64-
disk.img
    glance image-create --name "cirros" \
      --file cirros-0.4.0-x86_64-disk.img \
      --disk-format qcow2 --container-format bare \
      --visibility=public

- name: add centos image
  shell: |
    wget https://cloud.centos.org/centos/8/x86_64/images/CentOS-8-
GenericCloud-8.1.1911-20200113.3.x86_64.qcow2
    glance image-create --name "centos8" \
      --file CentOS-8-GenericCloud-8.1.1911-20200113.3.x86_64.qcow2 \
      --disk-format qcow2 --container-format bare \
      --visibility=public
    rm -f CentOS-8-GenericCloud-8.1.1911-20200113.3.x86_64.qcow2

- name: add ubuntu image
  shell: |
    wget https://cloud-images.ubuntu.com/bionic/current/bionic-server-
cloudimg-amd64.img
    glance image-create --name "ubuntu18" \
      --file bionic-server-cloudimg-amd64.img \
      --disk-format qcow2 --container-format bare \
      --visibility=public
    rm -f bionic-server-cloudimg-amd64.img

- name: add fedora coreos image
  shell: |
```

```
    wget https://builds.coreos.fedoraproject.org/prod/streams/stable/build
s/34.20210427.3.0/x86_64/fedora-coreos-34.20210427.3.0-
openstack.x86_64.qcow2.xz
    xz -d fedora-coreos-34.20210427.3.0-openstack.x86_64.qcow2.xz
    glance image-create --name "fedora34" \
      --file fedora-coreos-34.20210427.3.0-openstack.x86_64.qcow2 \
      --disk-format qcow2 --container-format bare \
      --property os_distro=fedora-coreos \
      --visibility=public
    rm -f fedora-coreos-34.20210427.3.0-openstack.x86_64.qcow2
```

Code for deploying Barbican:

```
- name: Create the barbican database
  shell: mysql -e "CREATE DATABASE barbican;"

# TODO update barbican password here
- name: Grant proper access to the barbican database
  shell: |
    mysql -
e "CREATE USER 'barbican'@'localhost' IDENTIFIED BY 'password';"
    mysql -
e "GRANT ALL PRIVILEGES ON barbican.* TO 'barbican'@'localhost';"
    mysql -e "CREATE USER 'barbican'@'%' IDENTIFIED BY 'password';"
    mysql -e "GRANT ALL PRIVILEGES ON barbican.* TO 'barbican'@'%';"
#   https://ma.ttias.be/mysql-8-removes-shorthand-creating-user-
permissions/

# TODO update barbican password here
- name: Create the barbican user
  shell: openstack user create --domain default --
password password barbican

- name: Add the admin role to the barbican user and service project
  shell: openstack role add --project service --user barbican admin

- name: Create the creator role
  shell: openstack role create creator

- name: Add the creator role to the barbican user
  shell: openstack role add --project service --user barbican creator

- name: Create the barbican service entity
```

```
  shell: openstack service create --name barbican --
description "Key Manager" key-manager

- name: Create the Container Infrastructure Management service API endpoin
ts
  shell: |
    openstack endpoint create --region RegionOne key-
manager public http://controller:9311
    openstack endpoint create --region RegionOne key-
manager internal http://controller:9311
    openstack endpoint create --region RegionOne key-
manager admin http://controller:9311

- name: Install the common and library packages
  apt:
    name: [barbican-api, barbican-keystone-listener, barbican-worker]
    state: latest

- name: Edit the /etc/barbican/barbican.conf file
  ini_file:
    path: /etc/barbican/barbican.conf
    owner: barbican
    group: barbican
    section: "{{ item.section }}"
    option: "{{ item.option }}"
    value: "{{ item.value }}"
  loop:
    - { section: "DEFAULT", option: "transport_url", value: 'rabbit://open
stack:password@localhost' }
    - { section: "DEFAULT", option: "sql_connection", value: "mysql+pymysq
l://barbican:password@localhost/barbican" }
    - { section: "keystone_authtoken", option: "www_authenticate_uri", val
ue: 'http://controller:5000' }
    - { section: "keystone_authtoken", option: "auth_url", value: 'http://
controller:5000' }
    - { section: "keystone_authtoken", option: "memcached_servers", value:
 'controller:11211' }
    - { section: "keystone_authtoken", option: "auth_type", value: 'passwo
rd' }
    - { section: "keystone_authtoken", option: "project_domain_name", valu
e: 'Default' }
    - { section: "keystone_authtoken", option: "user_domain_name", value:
'Default' }
    - { section: "keystone_authtoken", option: "project_name", value: 'ser
vice' }
```

```
    - { section: "keystone_authtoken", option: "username", value: 'barbica
n' }
    - { section: "keystone_authtoken", option: "password", value: 'passwor
d' }

- name: Populate the Key Manager service database
  shell: su -s /bin/sh -c "barbican-manage db upgrade" barbican

- name: Restart the Key Manager services
  service:
    name: "{{ item }}"
    state: restarted
  loop:
    - barbican-keystone-listener
    - barbican-worker
    - apache2
```

Code for deploying Neutron:

```
- name: Install and configure controller node
  include: controller-install.yml
  when: "'controller' in {{ group_names }}"

- name: Install and configure a compute node
  include: compute-install.yml
  when: "'compute' in {{ group_names }}"

- name: Add NAT iptable's role
  shell: |
    sysctl -w net.ipv4.ip_forward=1
    iptables -t nat -A POSTROUTING -
o {{ ansible_facts['default_ipv4']['interface'] }} -j MASQUERADE
  when: "'controller' in {{ group_names }}"

- name: verify successful launch of the neutron agents
  shell: openstack network agent list
  register: netreg

- name: print previous command output
  debug:
    var: netreg.stdout_lines
```

Code for deploying Heat:

```
- name: Create the heat database
  shell: mysql -e "CREATE DATABASE heat;"

# TODO update heat password here
- name: Grant proper access to the heat database
  shell: |
    mysql -e "CREATE USER 'heat'@'localhost' IDENTIFIED BY 'password';"
    mysql -e "GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'localhost';"
    mysql -e "CREATE USER 'heat'@'%' IDENTIFIED BY 'password';"
    mysql -e "GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'%';"
#   https://ma.ttias.be/mysql-8-removes-shorthand-creating-user-
permissions/

# TODO update heat password here
- name: Create the heat user
  shell: openstack user create --domain default --password password heat

- name: Add the admin role to the heat user and service project
  shell: openstack role add --project service --user heat admin

- name: Create the heat and heat-cfn service entities
  shell: |
    openstack service create --name heat --
description "Orchestration" orchestration
    openstack service create --name heat-cfn --
description "Orchestration"  cloudformation

- name: Create the Orchestration service API endpoints
  shell: |
    openstack endpoint create --
region RegionOne orchestration public http://controller:8004/v1/%\(tenant_
id\)s
    openstack endpoint create --
region RegionOne orchestration internal http://controller:8004/v1/%\(tenan
t_id\)s
    openstack endpoint create --
region RegionOne orchestration admin http://controller:8004/v1/%\(tenant_i
d\)s
```

```
    openstack endpoint create --
region RegionOne cloudformation public http://controller:8000/v1
    openstack endpoint create --
region RegionOne cloudformation internal http://controller:8000/v1
    openstack endpoint create --
region RegionOne cloudformation admin http://controller:8000/v1

- name: Create the heat domain that contains projects and users for stacks
  shell: openstack domain create --
description "Stack projects and users" heat

- name: Create the heat_domain_admin user to manage projects and users in
the heat domain
  shell: openstack user create --domain heat --
password password heat_domain_admin

- name: Add the admin role to the heat_domain_admin user in the heat domai
n
  shell: openstack role add --domain heat --user-domain heat --
user heat_domain_admin admin

- name: Create the heat_stack_owner role
  shell: openstack role create heat_stack_owner

- name: create demo project
  shell: openstack project create --domain heat --
description "Demo Project" demo

- name: create demo user
  shell: openstack user create --domain heat --password password demo

- name: Add the heat_stack_owner role to the demo project and user
  shell: openstack role add --project demo --user demo heat_stack_owner

- name: Create the heat_stack_user role
  shell: openstack role create heat_stack_user

- name:  Install the packages
  apt:
    name: [heat-api, heat-api-cfn, heat-engine]
    state: latest

- name: install missing pip packages
  shell: pip3 install python-zunclient python-vitrageclient
```

```
- name: Edit the /etc/heat/heat.conf file
  ini_file:
    path: /etc/heat/heat.conf
    owner: heat
    group: heat
    section: "{{ item.section }}"
    option: "{{ item.option }}"
    value: "{{ item.value }}"
  loop:
    - { section: "database", option: "connection", value: 'mysql+pymysql:/
/heat:password@localhost/heat' }
    - { section: "keystone_authtoken", option: "www_authenticate_uri", val
ue: 'http://controller:5000' }
    - { section: "keystone_authtoken", option: "auth_url", value: 'http://
controller:5000' }
    - { section: "keystone_authtoken", option: "memcached_servers", value:
 'controller:11211' }
    - { section: "keystone_authtoken", option: "auth_type", value: 'passwo
rd' }
    - { section: "keystone_authtoken", option: "project_domain_name", valu
e: 'Default' }
    - { section: "keystone_authtoken", option: "user_domain_name", value:
'Default' }
    - { section: "keystone_authtoken", option: "project_name", value: 'ser
vice' }
    - { section: "keystone_authtoken", option: "username", value: 'heat' }
    - { section: "keystone_authtoken", option: "password", value: 'passwor
d' }
    - { section: "DEFAULT", option: "transport_url", value: 'rabbit://open
stack:password@localhost' }
    - { section: "DEFAULT", option: "heat_metadata_server_url", value: "ht
tp://{{ controller_ip }}:8000"}
    - { section: "DEFAULT", option: "heat_waitcondition_server_url", value
: "http://{{ controller_ip }}:8000/v1/waitcondition" }
    - { section: "DEFAULT", option: "stack_domain_admin", value: 'heat_dom
ain_admin' }
    - { section: "DEFAULT", option: "stack_domain_admin_password", value:
'password' }
    - { section: "DEFAULT", option: "stack_user_domain_name", value: 'heat
' }
    - { section: "trustee", option: "auth_type", value: 'password' }
    - { section: "trustee", option: "auth_url", value: 'http://controller:
5000' }
    - { section: "trustee", option: "username", value: 'heat' }
    - { section: "trustee", option: "password", value: 'password' }
```

```
    - { section: "trustee", option: "user_domain_name", value: 'Default' }
    - { section: "clients_keystone", option: "auth_uri", value: 'http://co
ntroller:5000' }

- name: Populate the Orchestration database
  shell: su -s /bin/sh -c "heat-manage db_sync" heat

- name: Restart the Orchestration services
  service:
    name: "{{ item }}"
    state: restarted
  loop:
    - heat-api
    - heat-api-cfn
    - heat-engine

- name: List service components to verify successful launch and registrati
on of each process
  register: heatreg
  shell: openstack orchestration service list

- name: print previous command output
  debug:
    var: heatreg.stdout_lines
```

Code for deploying Placement:

```
- name: Create the placement database
  shell: mysql -e "CREATE DATABASE placement;"

# TODO update placement password here
- name: Grant proper access to the placement database
  shell: |
    mysql -
e "CREATE USER 'placement'@'localhost' IDENTIFIED BY 'password';"
    mysql -
e "GRANT ALL PRIVILEGES ON placement.* TO 'placement'@'localhost';"
    mysql -e "CREATE USER 'placement'@'%' IDENTIFIED BY 'password';"
    mysql -e "GRANT ALL PRIVILEGES ON placement.* TO 'placement'@'%';"
#   https://ma.ttias.be/mysql-8-removes-shorthand-creating-user-
permissions/

# TODO update placement password here
- name: Create the placement user
```

```
  shell: openstack user create --domain default --
password password placement

- name: Add the admin role to the placement user and service project
  shell: openstack role add --project service --user placement admin

- name: Create the placement service entity
  shell: openstack service create --name placement --
description "Placement API" placement

- name: Create the placement service API endpoints
  shell: |
    openstack endpoint create --
region RegionOne placement public http://controller:8778
    openstack endpoint create --
region RegionOne placement internal http://controller:8778
    openstack endpoint create --
region RegionOne placement admin http://controller:8778

- name:  install placement
  apt:
    name: placement-api
    state: latest

# TODO update placement password here
- name: Edit the /etc/placement/placement.conf file
  ini_file:
    path: /etc/placement/placement.conf
    owner: placement
    group: placement
    section: "{{ item.section }}"
    option: "{{ item.option }}"
    value: "{{ item.value }}"
  loop:
    - { section: "placement_database", option: "connection", value: 'mysql
+pymysql://placement:password@localhost/placement' }
    - { section: "keystone_authtoken", option: "www_authenticate_uri", val
ue: 'http://controller:5000' }
    - { section: "keystone_authtoken", option: "auth_url", value: 'http://
controller:5000' }
    - { section: "keystone_authtoken", option: "memcached_servers", value:
 'controller:11211' }
    - { section: "keystone_authtoken", option: "auth_type", value: 'passwo
rd' }
```

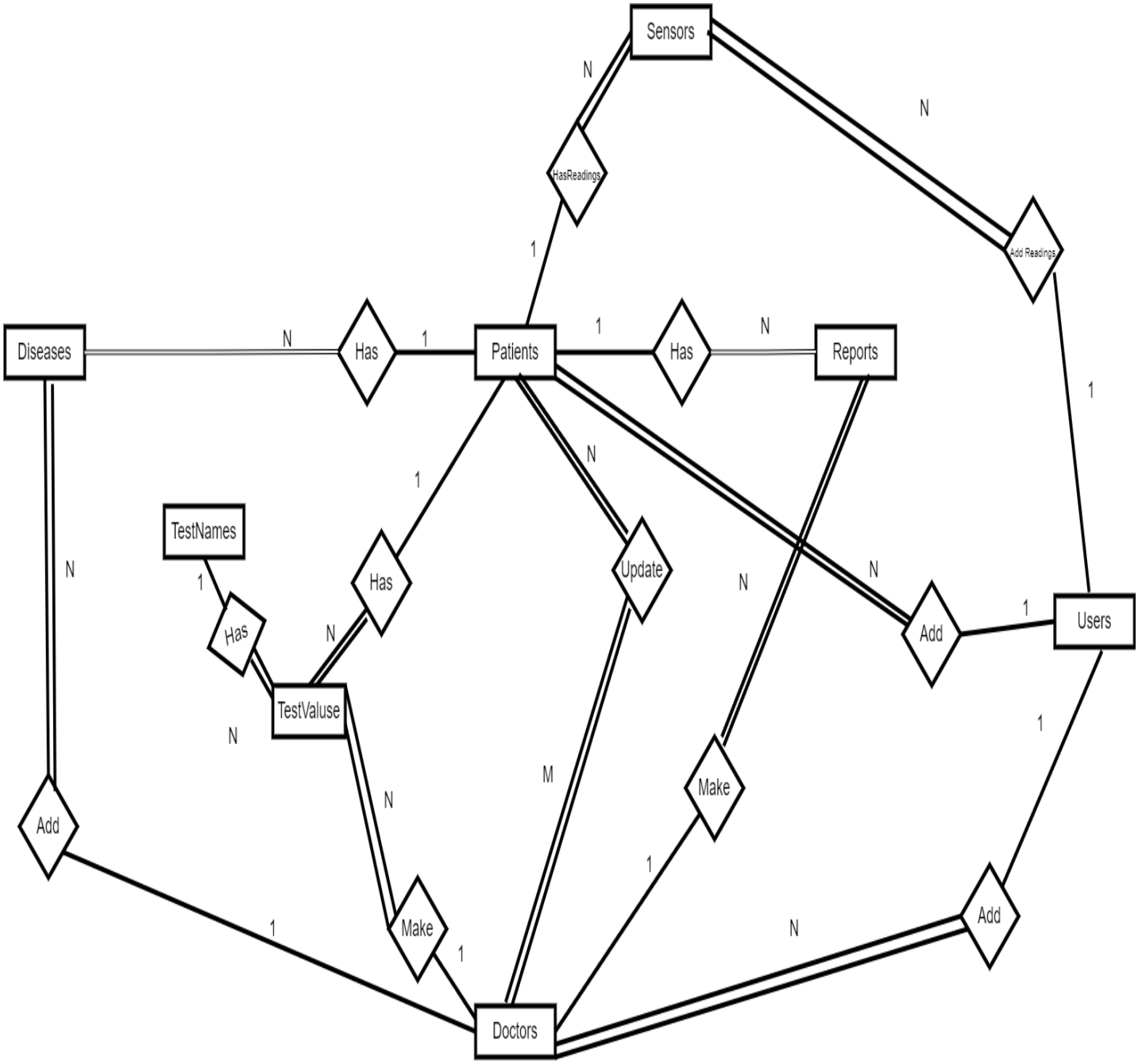
```
    - { section: "keystone_authtoken", option: "project_domain_name", valu
e: 'Default' }
    - { section: "keystone_authtoken", option: "user_domain_name", value:
'Default' }
    - { section: "keystone_authtoken", option: "project_name", value: 'ser
vice' }
    - { section: "keystone_authtoken", option: "username", value: 'placeme
nt' }
    - { section: "keystone_authtoken", option: "password", value: 'passwor
d' }
    - { section: "api", option: "auth_strategy", value: 'keystone' }

- name: Populate the placement database
  shell: su -s /bin/sh -c "placement-manage db sync" placement

- name: Restart service apache2
  service:
    name: apache2
    state: restarted

- name: Install the osc-placement plugin
  shell: pip3 install osc-placement

- name: List available resource classes and traits
  shell: |
    openstack --os-placement-api-version 1.2 resource class list --sort-
column name
    openstack --os-placement-api-version 1.6 trait list --sort-column name
```

Chapter 3
Application

3.1 Web Backend

3.1.1 Database Design

3.1.1.1 Er Diagram



3.1.1.2 Tables

- Patients table:

Server: 127.0.0.1 » Database: clocare » Table: patients

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracking | Triggers

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	national_id	bigint(20)			No	None			Change Drop More
3	name	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
4	password	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
5	phone_number	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
6	address	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
7	arriving_date	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
8	api_token	varchar(100)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
9	birth_date	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
10	age	int(11)			Yes	NULL			Change Drop More
11	height	int(11)			Yes	NULL			Change Drop More
12	weight	int(11)			Yes	NULL			Change Drop More
13	gender	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
14	bloodType	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
15	profile_photo_path	text	utf8mb4_unicode_ci		Yes	NULL			Change Drop More

Server: 127.0.0.1 » Database: clocare » Table: patients

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracking | Triggers

13	gender	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
14	bloodType	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
15	profile_photo_path	text	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
16	statues	tinyint(1)			No	0			Change Drop More
17	hospital_name	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
18	medicines	longtext	utf8mb4_unicode_ci		Yes	NULL		(DC2Type:json)	Change Drop More
19	education	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
20	marital	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
21	insurance	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
22	gen_health	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
23	smoker	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
24	days_active	int(11)			Yes	NULL			Change Drop More
25	bmi	double			Yes	NULL			Change Drop More
26	waist	int(11)			Yes	NULL			Change Drop More
27	drinks_day	int(11)			Yes	NULL			Change Drop More
28	income	int(11)			Yes	NULL			Change Drop More
29	created_at	timestamp			Yes	NULL			Change Drop More
30	updated_at	timestamp			Yes	NULL			Change Drop More

- Doctors:

Server: 127.0.0.1 » Database: clocare » Table: doctors

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracking | Triggers

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	national_id	bigint(20)			No	None			Change Drop More
3	name	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
4	email	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
5	password	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
6	profile_photo_path	text	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
7	specialization	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
8	phone_number	int(11)			Yes	NULL			Change Drop More
9	work_at	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
10	api_token	varchar(100)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
11	created_at	timestamp			Yes	NULL			Change Drop More
12	updated_at	timestamp			Yes	NULL			Change Drop More

- Diseases table:

Server: 127.0.0.1 » Database: clocare » Table: diseaes

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracking | Triggers

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	doctor_id	bigint(20)		UNSIGNED	No	None			Change Drop More
3	patient_id	bigint(20)		UNSIGNED	No	None			Change Drop More
4	asthma	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
5	chf	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
6	cad	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
7	mi	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
8	cva	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
9	copd	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
10	cancer	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
11	hypertension	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
12	diabetes	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
13	pulse	int(11)			Yes	NULL			Change Drop More
14	sys_bp	double(8,2)			Yes	NULL			Change Drop More
15	dia_bp	double(8,2)			Yes	NULL			Change Drop More
16	created_at	timestamp			Yes	NULL			Change Drop More
17	updated_at	timestamp			Yes	NULL			Change Drop More

- Report table:

Server: 127.0.0.1 » Database: clocare » Table: reports

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracking | Triggers

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	doctor_id	bigint(20)		UNSIGNED	No	None			Change Drop More
3	patient_id	bigint(20)		UNSIGNED	No	None			Change Drop More
4	reports_photo_path	text	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
5	diagnose	text	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
6	medicine	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
7	traits	text	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
8	department	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
9	comments	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
10	arriving_date	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
11	discharge_date	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
12	created_at	timestamp			Yes	NULL			Change Drop More
13	updated_at	timestamp			Yes	NULL			Change Drop More

- Sensors table:

Server: 127.0.0.1 » Database: clocare » Table: sensors

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracking | Triggers

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	patient_id	bigint(20)		UNSIGNED	No	None			Change Drop More
3	spo2	int(11)			No	None			Change Drop More
4	heartRate	int(11)			No	None			Change Drop More
5	temp	double			No	None			Change Drop More
6	created_at	timestamp			Yes	NULL			Change Drop More
7	updated_at	timestamp			Yes	NULL			Change Drop More

- Test Names tables:

Server: 127.0.0.1 » Database: clocare » Table: test_names

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracking | Triggers

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	name	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
3	created_at	timestamp			Yes	NULL			Change Drop More
4	updated_at	timestamp			Yes	NULL			Change Drop More

- Test Values:

Server: 127.0.0.1 » Database: clocare » Table: test_values

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracking | Triggers

15	potassium	double(8,2)		Yes	NULL	Change	Drop	More
16	chloride	double(8,2)		Yes	NULL	Change	Drop	More
17	bicrab	double(8,2)		Yes	NULL	Change	Drop	More
18	ca	double(8,2)		Yes	NULL	Change	Drop	More
19	phos	double(8,2)		Yes	NULL	Change	Drop	More
20	t_bilirubin	double(8,2)		Yes	NULL	Change	Drop	More
21	alb	double(8,2)		Yes	NULL	Change	Drop	More
22	t_protein	double(8,2)		Yes	NULL	Change	Drop	More
23	glob	double(8,2)		Yes	NULL	Change	Drop	More
24	glucose	double(8,2)		Yes	NULL	Change	Drop	More
25	glucose1	double(8,2)		Yes	NULL	Change	Drop	More
26	alb_cr_ratio	double(8,2)		Yes	NULL	Change	Drop	More
27	trigs	double(8,2)		Yes	NULL	Change	Drop	More
28	t_chol	double(8,2)		Yes	NULL	Change	Drop	More
29	hdl	double(8,2)		Yes	NULL	Change	Drop	More
30	ldl_chol	double(8,2)		Yes	NULL	Change	Drop	More
31	a1c	double(8,2)		Yes	NULL	Change	Drop	More

Activate Windows

Server: 127.0.0.1 » Database: clocare » Table: test_values

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracking | Triggers

29	hdl	double(8,2)		Yes	NULL	Change	Drop	More
30	ldl_chol	double(8,2)		Yes	NULL	Change	Drop	More
31	a1c	double(8,2)		Yes	NULL	Change	Drop	More
32	insulin	double(8,2)		Yes	NULL	Change	Drop	More
33	iron	double(8,2)		Yes	NULL	Change	Drop	More
34	u_acid	double(8,2)		Yes	NULL	Change	Drop	More
35	s_cotinine	double(8,2)		Yes	NULL	Change	Drop	More
36	cpk	double(8,2)		Yes	NULL	Change	Drop	More
37	ldh	double(8,2)		Yes	NULL	Change	Drop	More
38	fvc	double(8,2)		Yes	NULL	Change	Drop	More
39	fev1	double(8,2)		Yes	NULL	Change	Drop	More
40	fev1_fvc_ratio	double(8,2)		Yes	NULL	Change	Drop	More
41	memory	varchar(255)	utf8mb4_unicode_ci	Yes	NULL	Change	Drop	More
42	created_at	timestamp		Yes	NULL	Change	Drop	More
43	updated_at	timestamp		Yes	NULL	Change	Drop	More

Check all *With selected:* Browse | Change | Drop | Primary | Unique | Index | Spatial | Fulltext

Add to central columns | Remove from central columns

Activate Windows
Go to Settings to activate Windows.

- Admins and user's table:

Server: 127.0.0.1 » Database: clocare » Table: users

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracking | Triggers

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	national_id	bigint(20)			No	None			Change Drop More
3	name	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
4	email	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
5	email_verified_at	timestamp			Yes	NULL			Change Drop More
6	password	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
7	two_factor_secret	text	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
8	two_factor_recovery_codes	text	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
9	api_token	varchar(100)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
10	is_admin	tinyint(1)			Yes	NULL			Change Drop More
11	specialization	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
12	work_at	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
13	phone_number	bigint(20)			Yes	NULL			Change Drop More
14	address	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
15	remember_token	varchar(100)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
16	current_team_id	bigint(20)		UNSIGNED	Yes	NULL			Change Drop More
17	profile_photo_path	text	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
18	created_at	timestamp			Yes	NULL			Change Drop More
19	updated_at	timestamp			Yes	NULL			Change Drop More

Activate Windows
Go to Settings to activate Windows.

3.1.2 Tools

3.1.2.1 Php

PHP is a server-side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites. It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.

- Common use of PHP:
  - PHP performs system functions, i.e., from files on a system it can create, open, read, write, and close them.
  - PHP can handle forms, i.e., gather data from files, save data to a file, through email you can send data, return data to the user.
  - You add, delete, modify elements within your database through PHP.
  - Access cookies variables and set cookies.
  - Using PHP, you can restrict users to access some pages of your website.
  - It can encrypt data.
- Characteristics of PHP:
  - Five important characteristics make PHP's practical nature possible.
  - Simplicity
  - Efficiency
  - Security
  - Flexibility
  - Familiarity

3.1.2.2 LARAVEL

Laravel is primarily used for building custom web apps using PHP. It is a web framework that handles many things that are annoying to build yourself, such as routing, templating HTML, and authentication.

Laravel is entirely server-side, due to running on PHP, and focuses heavily on data manipulation and sticking to a Model-View-Controller design. A framework like React might put most of its attention on user interaction and shiny features, but Laravel simply presents a solid foundation for you to build off—and does it right.

Laravel is one of the best PHP web frameworks, but there are many other frameworks in different languages. Rails is another server-side rendered framework, like Laravel,

but based on Ruby. React, Vue, and Angular are all client-side JavaScript frameworks but can be configured to render server-side as well.

Alternatively, if an application leans towards a blog style with multiple text-based posts, we could base it off WordPress, which also runs on PHP. But Laravel does not force us to use features we do not want; it just gives us the tools to build something like WordPress by yourself.

Laravel uses a design pattern called Model-View-Controller, or MVC. The "Model" represents the shape of the data your application operates on. If we have a table of users, each with a list of posts they have made, that is your model.

The "Controller" interacts with this model. If a user requests to see their posts page, the controller talks to the model (often just the database) and retrieves the info. If the user wants to make a new post, the controller updates the model. The controller contains most of the logic for your application.

The controller uses that info to construct a "View." The view is a template with which the model can be plugged into and displayed, and it can be manipulated by the controller. The view is all your application's HTML components.

Laravel uses this structure to power custom apps. It uses the Blade templating engine, which allows HTML to be broken into pieces and managed by the controller. It all starts with routes, defined in routes/web.php, that handle HTTP requests based on the location being requested

For example, the following function would run if a user requested https://yoursite.com/greeting:

```
Route::get ('/greeting', function () {  return view ('greeting', ['name' => 'James']);});
```

This route executes a function that returns a view from resources/views/. The view has been passed data (the name variable), which it can use inside the markup:

```
<! -- View stored in resources/views/greeting.blade.php -->

<html>   <body>      <h1>Hello, {{$name}} </h1>   </body></html>
```

This is as simple as it gets, but a lot can happen in between the request and the return of a view. Laravel supports middleware, which will run before the request is handled. You could use this to lock down certain pages by checking if a user is authenticated before a request is handled.

Instead of showing a view directly, you can also pass the request off to a controller, which can handle more complex logic before returning some resource (often a view). You can read more about the inner workings of the Laravel framework in their docs.

3.1.2.3 VSCODE

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity)

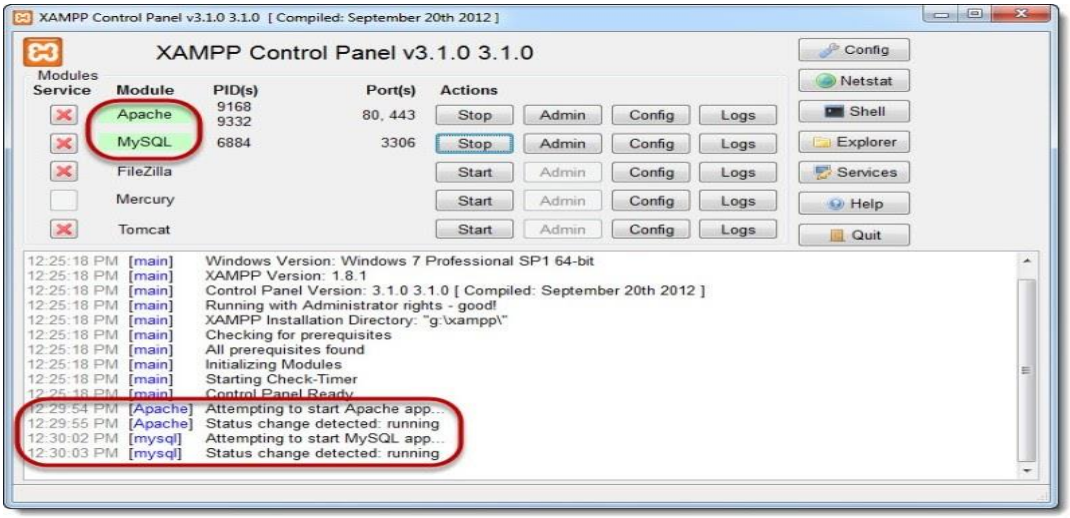
3.1.2.4 XAMPP

XAMPP is an abbreviation for cross-platform, Apache, MySQL, PHP and Perl, and it allows you to build Website offline, on a local web server on your computer. This simple and lightweight solution works on Windows, Linux, and Mac – hence the "cross-platform" part.

it provides two essential components for its installation – Apache, which is used to create the local server, and MySQL which you can use as a database for your website.

So, to create my local database "CloCare":

- run Apache and MySQL servers.
- reach'' phpMyAdmin page'' by typing "http://localhost/phpmyadmin/" (without the quotation marks) into my browser.
- Find "Databases" link on the top of the page, and right under it. Click "Create Database". I enter the desired name 'Clo Care' and click "Create."



3.1.2.5 Postman

Postman is an interactive and automatic tool for verifying the APIs of a project. Postman is a Google Chrome app for interacting with HTTP APIs. It presents a friendly GUI for constructing requests and reading responses. It works on the backend, and makes sure that each API is working as intended.

In Postman, I create a request, and Postman looks at the response to make sure it has the element I want in it. As it is an automation tool, it drastically improves testing time and quality of the project. It helps in the early detection of bugs that might sprout at later stages and cause more damage to the system. Postman is the way to streamline the process of API testing. All APIs that I create and deploy first rigorously go through Postman so that any major or show stopper bugs are identified on time and fewer bugs leak through to later stages.

3.1.3 system features

3.1.3.1 Authentication and authorization

- URLS:
  - A middleware is used to protect the routes of (web) dashboard so only admins can have the access to it.
  - This middleware is called (is_admin) and its implementation is

```
app > Http > Middleware > IsAdmin.php > ...
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle(Request $request, Closure $next)
    {
    //     if (Auth::check() &&  Auth::user()->is_admin == 1) {
    //         return $next($request);
    //    }else{
    //     return redirect()->back();

    //    }
    if (Auth()->check() && Auth()->user()->is_admin == 1)
    {
    return $next($request);
        }else{
         return redirect()->back();

        }
}
}
```

- loginAdmin function is created in the controller file to check every request (user)trying to access the dashboard as only the admins have that access and no one else.

```
app > Http > Controllers > Controller.php > ...
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class Controller extends BaseController
{
    use AuthorizesRequests, DispatchesJobs, ValidatesRequests;

public function loginAdmin(Request $request ){
    if(Auth::attempt(
        [
            'email'=>$request['email'],
            'password'=>$request['password'],
            'is_admin'=>1
        ]
    )){
        return redirect('/dashboard');
    }
    else{
        echo"<h2>you are not allowed to get to this page</h2> ";
    }
}

}
```

- All routes that are related to the dashboard are being grouped under the protection of (is_admin)middleware that has been previously created

```
routes > web.php > #Function#69df866f
12  | contains the "web" middleware group. Now create something great!
13  |
14  */
15  Route::post('/adminlogin','App\Http\Controllers\Controller@loginAdmin')->name('adminlogin');
16
17  ////Route::middleware(['auth:sanctum', 'verified'])->group(function(){
18      Route::group(['middleware' => 'is_admin'], function (){
19          Route::get('/dashboard',function(){
20              return view('dashboardReal');
21          })->name('dashboard');
22
23          Route::resource('Doctors',App\Http\Controllers\Dashboard\Doctors::class);
24          Route::resource('Patients',App\Http\Controllers\Dashboard\Patients::class);
25          Route::resource('Employee',App\Http\Controllers\Dashboard\Employee::class);
26
27
28          Route::get('/getPatientsOut','App\Http\Controllers\Dashboard\Patients@indexOut')->name('getPatientsOut');
29          Route::get('/getPatientsIn','App\Http\Controllers\Dashboard\Patients@indexIn')->name('getPatientsIn');
30
31          Route::get('/showSensorReadingsForPatient/{id}','App\Http\Controllers\Dashboard\Patients@showSensorReadingsForPatie
32          Route::get('/showTestValues/{test_id}/{patient_id}','App\Http\Controllers\Dashboard\Patients@showTestValues')->name
33          Route::get('/showReports/{id}','App\Http\Controllers\Dashboard\Patients@showReports')->name('showReports');
34
35      });
```

- APIs:
  As the APIS are mainly concerned with the mobile application that have two different users: doctors and patients, a mechanism called guards are being used to protect doctors' routes and patients' routes from each other and prevent overlapping by checking api_tokens in each table.

```
config > auth.php
35          | Supported: "session", "token"
36          |
37          */
38
39          'guards' => [
40              'web' => [
41                  'driver' => 'session',
42                  'provider' => 'users',
43              ],
44              'api' => [
45                  'driver' => 'token',
46                  'provider' => 'users',
47                  'hash' => false,
48              ],
49
50
51              'doctor' => [
52                  'driver' => 'token',
53                  'provider' => 'doctors',
54                  'hash' => false,
55              ],
56              'patient' => [
57                  'driver' => 'token',
58                  'provider' => 'patients',
59                  'hash' => false,
60              ],
61          ],
62
```

- Login and register functions:
  - URLS:
    Laravel 8 has new feature called (Jetstream) that automatically generate all web Auth functions and their views.
  - APIS:
    Mobile application does not have any register methods as doctors and patients are added by the admins through dashboard. On the other hand, login functions are implemented like this:

```
app > Http > Controllers > Api > Doctors > AuthController.php > ...
17  {
18
19      public function login (Request $request) {
20          try{
21              $validator = Validator::make($request->all(), [
22                  'national_id' => 'required|string',
23                  'password' => 'required|string|min:5',
24              ]);
25              if ($validator->fails())
26              {
27                  return response([
28                      "message"=>'The given data was invalid.',
29                      "errors"=>$validator->errors()
30
31                  ], 422);
32              }
33               $user = Doctors::where('national_id', $request->national_id)->first();
34              if ($user) {
35                  if (Hash::check( $request->password,$user->password )) {
36                      $token = $user->api_token;
37                      return [ 'doctor'=>new Doctor($user)];
38                  } else {
39
40                      return response()->json( ["message" => "Password mismatch"],422);
41
42                  }
43              } else {
44
45                  return response()->json(["message" =>'User does not exist'],422);
46              }
47          }catch(Exception $e){
48              return response()->json(['errors' => $e->getMessage()], 500);
49          }
50
51      }
```

3.1.3.2 Interacting with monitor device(sensors)

The interacting with the monitor device is done mainly through two Apies: The first one is for the device to log in the system using the patient's account that is being monitored.

Api:127.0.0.1:8000/loginPatient(post request)

The second one is to take all sensor readings from the monitor device of that particular patient that has been previously logged in and store them in the database.

Api:127.0.0.1:8000/Sensor (post request)

Only the last ten sensor readings will be kept in the database to save storage.

```
app > Http > Controllers > Api > Patients > SensorsController.php > SensorsController > store
27          *
28          * @param  \Illuminate\Http\Request  $request
29          * @return \Illuminate\Http\Response
30          */
31         public function store(Request $request)
32         {
33             $patient = Patients::find(Auth()->user()->id);
34             $sensor = new Sensors();
35             $sensor->spo2 = $request['spo2'];
36             $sensor->heartRate = $request['heartRate'];
37             $sensor->temp = $request['temp'];
38             $patient->sensors()->save($sensor);
39             $ids = [];
40             $count = $patient->sensors()->count();
41
42             $deleteUs = $patient->sensors()->latest()->take($count)->skip(10)->get();
43
44             foreach ($deleteUs as $deleteMe) {
45                 $ids[] = $deleteMe->id;
46             }
47
48             Sensors::destroy($ids);
49             return response()->json($patient->sensors()->get());
50         }
51
```

3.1.3.3 Doctor functions

The main purpose of the app is to facilitates some of routine functions for doctors and here are some of these functions:

- Have the access to all patient's data that are in the hospital
- Show data of a particular patient
- Update data of that patient.
- Delete patient from the system

all the previous functions are being done through this Api resource:

```
Route::resource('/DRpatient','App\Http\Controllers\Api\Doctors\DoctorController');
```

Update function implementation is:

```
> Http > Controllers > Api > Doctors > DoctorController.php > DoctorController > update
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(PatientsUpdate $request, $id)
    {
        try{
            $patient=Patients::find($id);
            $validatedData=$request->validated();
            $patient->fill($validatedData);
            if($request->hasFile('profile_photo_path')){

            if($patient->profile_photo_path){
                Storage::delete($patient->profile_photo_path);
                $path=$request->file('profile_photo_path')->store('patients');
            }
             $patient->profile_photo_path=$path;
            }
             $patient->save();
             return response()->json(["Updated patient"=>$patient],200);
        }
        catch(Exception $e){
            return response()->json(['errors' => $e->getMessage()], 500);
        }


    }
```

Add reports for every patient:

Api:127.0.0.1:8000/addreports/{id} (post request)

Add report function implementation is: in that report doctor add comments, traits, medicines, coming data, discharge data for every patient.

```
app > Http > Controllers > Api > Doctors > DoctorController.php > DoctorController > update
    public function AddReports( Addreports $request ,$id){

            try{
        $patient= Patients::find($id);
        // $validatedData=$request->validated();
        if($patient){
            $validatedData=$request->validated();
            $report =new Reports;
            if($request->hasFile('reports_photo_path')){
                $photoreport=$request->file('reports_photo_path')->store('reports');
                $report->reports_photo_path=$photoreport;

            }
            $report->fill($validatedData);
            $report->doctor_id=Auth::user()->id;
            $patient->statues=1;
            // $patient->save();
            // $patient->report()->save($report);

            $doctor=Doctors::where('id',Auth::user()->id);
                return response()->json([
                    'message'=>' report has been added successfuly',
                    'patient ID'=>$patient->id
                ,   ' reports for the same patient '=>
               $patient->where('id',$id)->with('report','report.doctorMadeReport')->get()
                    // $report->with('doctor')->where('patient_id',$patient->id)->get()
               //  ,'doctor that make the report '=>$patient->hasReports()->get()
                ],200);
                }else{
                    return response()->json(['message'=>'patient is not found so you can not add reports'],404);
                }


        }catch(Exception $e){
         return response()->json(['errors' => $e->getMessage()], 500);
```

Add test results for every patient:

Api:127.0.0.1:8000/addTest/{test_id}/{patient_id}

function implementation:

```
public function addtest(Test $request,$test_id,$patient_id){

    try{
        $patient= Patients::find($patient_id);
        // $validatedData=$request->validated();
        if($patient){
            $validatedData=$request->validated();
             $test =new TestValue;
            $test->fill($validatedData);
            $test->test_id=$test_id;
             $test->doctor_id=Auth::user()->id;
            $patient->test()->save($test);

                return response()->json([
                    'message'=>' test has been added successfuly',
                    'patient ID'=>$patient->id
                ,   ' tests for tha same patient'=>$patient->where('id',$patient_id)->with('test')->get()
               //  ,'doctor that make the report '=>$patient->hasReports()->get()
                ],200);
                }else{
                    return response()->json(['message'=>'patient is not found so you can not add reports'],404);
                }


       }catch(Exception $e){
        return response()->json(['errors' => $e->getMessage()], 500);
       }
```

Show medical history of the patient:

Api:127.0.0.1:8000 /showMedicalHistory/{id} (get request)

Function implementation:

```
app > Http > Controllers > Api > Doctors > DoctorController.php > DoctorController > update
                return response()->json(['errors' => $e->getMessage()], 500);
            }



        }

            public function showMedicalHistory ($id){
                try{

                    $patient= Patients::find($id);
                    if($patient){
                        return response()->json([
                            // 'all reports this patient has '=>$report
                     'medical_history ' => $patient->where('id',$id)->with('report','doctorMadeReport')->with('disease')->with('test')->get(),
                    //    'sdfs'=> $patient->makeReports()->limit(1)->get()
                        ]
                        ,200);

                    }else{
                        return response()->json(['message'=>'patient is not found'],404);
                    }
                }catch(Exception $e){
                    return response()->json(['errors' => $e->getMessage()], 500);
                }



            }
```

Search for specific patient:

Api:127.0.0.1:8000/searchPatient (get request)

Function implementation:

```
app > Http > Controllers > Api > Doctors > DoctorController.php > DoctorController > update
222
223
224     public function searchPatient(Request $request){
225         try{
226       $word = $request->get('search');
227      $patients= Patients::where('national_id', 'LIKE', "%{$word}%")->with('disease')->with('test')->with('report','report.doctorMadeReport')->get();
228       if($patients){
229           // return response()->json(['patients found are '=>
230           // $patients->where('id',$patients->id)->with('report')->with('disease')->with('test')->get()],200);
231           return  response()->json(['patients'=> $patients],200);
232       }else{
233           return response()->json(['message'=>'no patient found'],404);
234       }
235       }catch(Exception $e){
236       return response()->json(['errors' => $e->getMessage()], 500);
237        }
238
239     }
240
```

3.1.3.4 Patient functions

Patients only have access to their medical history as it was shown previously.

3.1.3.5 Notification

Firebase Messaging (FCM) is used to send notifications to doctors' phones in case of abnormal readings received from monitoring devices. Based on the following logic:

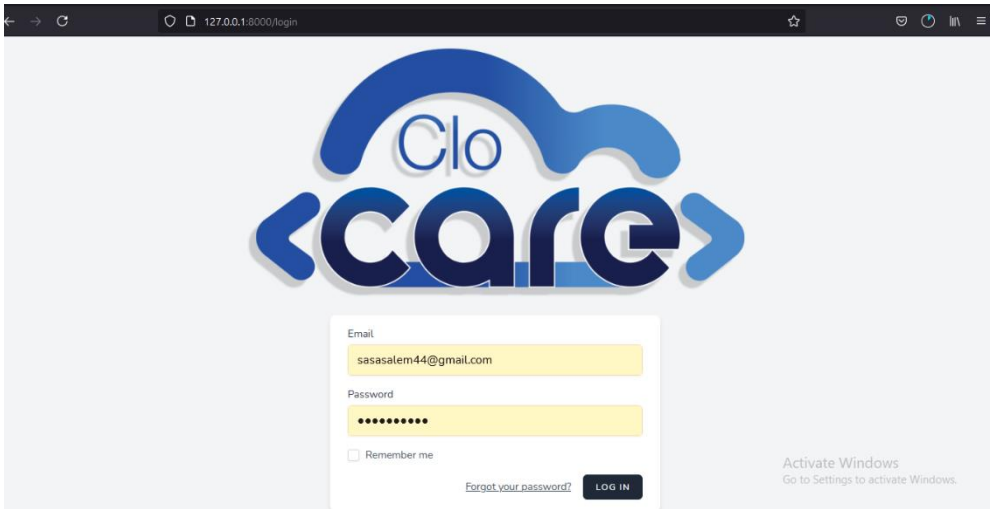
```
app > Http > Controllers > Api > Patients > SensorsController.php > SensorsController > sendNotification
23      }
24
25      private function sendNotification(Request $request)
26
27      {
28          $body = " ";
29          $firebaseToken = Doctors::whereNotNull('device_token')->pluck('device_token')->all();
30
31
32          $SERVER_API_KEY = 'AAAAXV426tk:APA91bHtqNVU4U63DTWNk7Lj12XyPWLWVzhH8o3sKjghAxJaXxLTmtI
33          if ($request['spo2'] >= 120) {
34              $info = "spo for this patient is very high";
35              $body += $info;
36          } elseif ($request['spo2'] >= 75) {
37              $info = "spo for this patient is very low";
38              $body += $info;
39          }
40
41          if ($request['heartRate'] >= 190) {
42              $info2 = "heartRate for this patient is very high";
43              $body += $info2;
44          } elseif ($request['heartRate'] >= 60) {
45              $info2 = "heartRate for this patient is very low";
46              $body += $info2;
47          }
48
49          if ($request['temp'] >= 190) {
50              $info3 = "temprature for this patient is very high";
51              $body += $info3;
52          } elseif ($request['temp'] >= 60) {
53              $info3 = "temprature for this patient is very low";
54              $body += $info3;
55          }
```

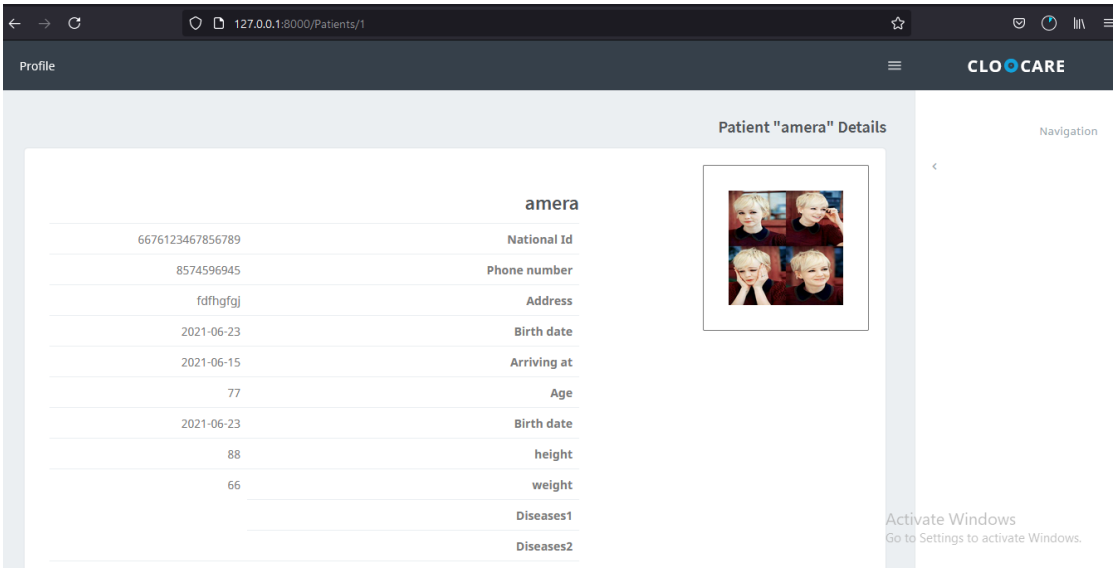
app > Http > Controllers > Api > Patients > SensorsController.php > SensorsController > sendNotification

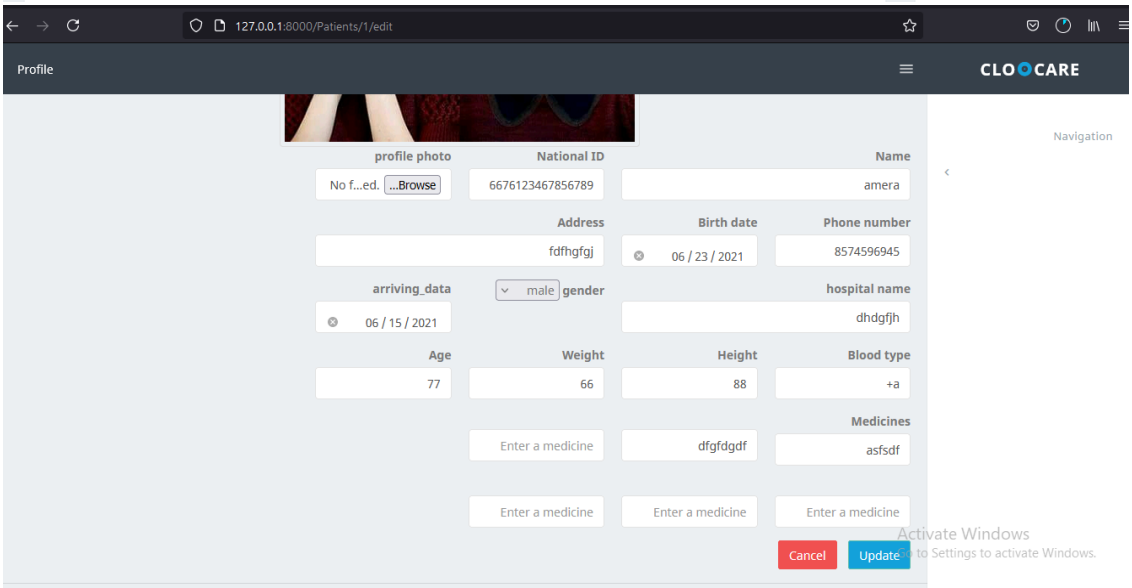
```php
        if ($request['temp'] >= 190) {
            $info3 = "temprature for this patient is very high";
            $body += $info3;
        } elseif ($request['temp'] >= 60) {
            $info3 = "temprature for this patient is very low";
            $body += $info3;
        }

        $data = [
            "registration_ids" => $firebaseToken,
            "notification" => [
                "title" => "attention!!!",
                "body" => $body
            ] ,   "data" => [
                "click_action" => "FLUTTER_NOTIFICATION_CLICK",
            ]

        ];
        $dataString = json_encode($data);
        $headers = [
            'Authorization: key=' . $SERVER_API_KEY,
            'Content-Type: application/json',
        ];

        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, 'https://fcm.googleapis.com/fcm/send');
        curl_setopt($ch, CURLOPT_POST, true);
        curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($ch, CURLOPT_POSTFIELDS, $dataString);
        $response = curl_exec($ch);
    }
```

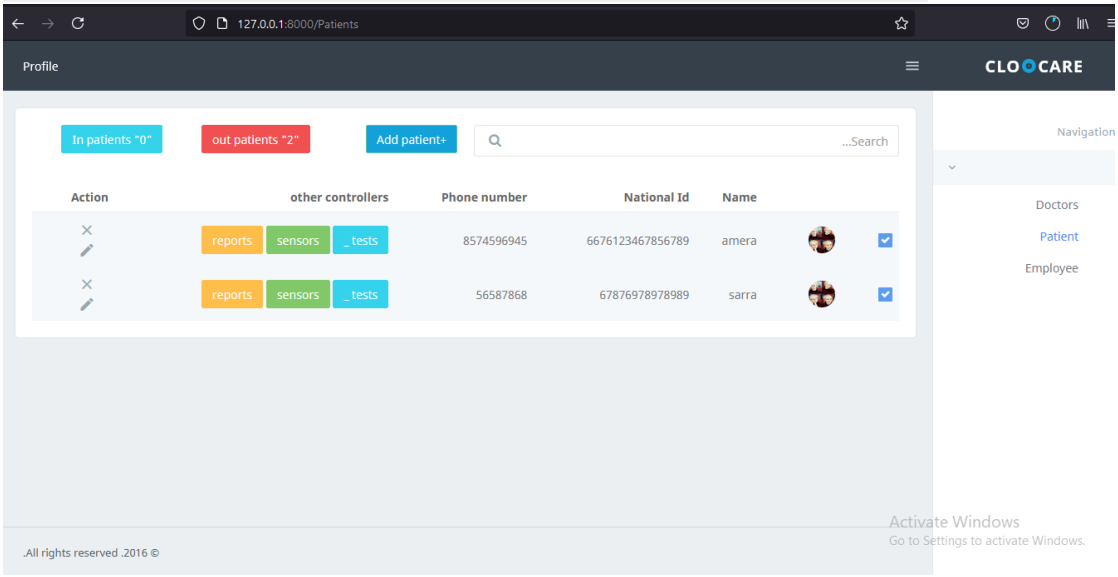
3.1.3.6 Admin Panel

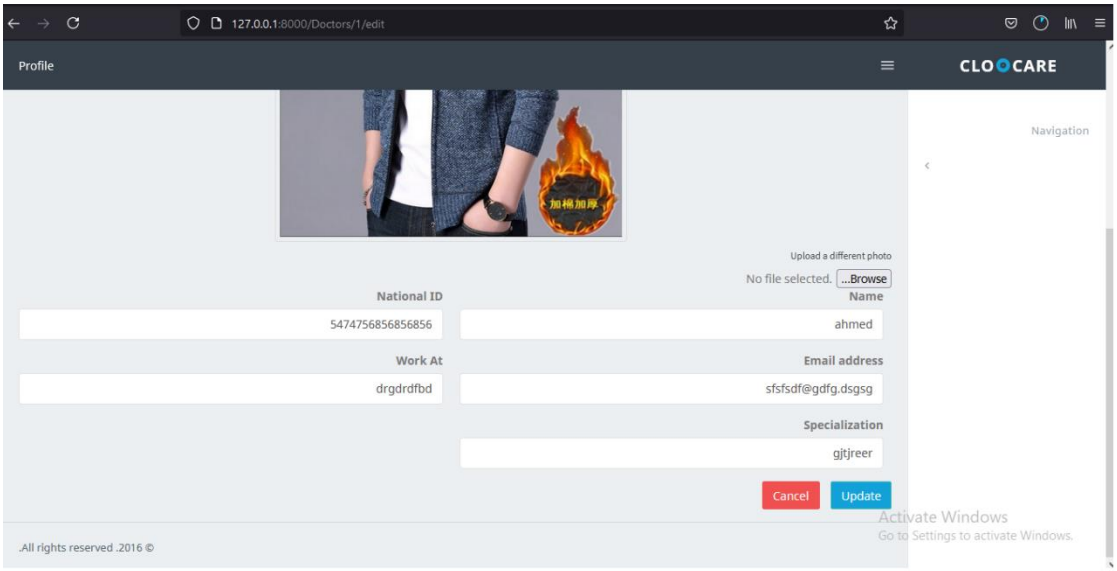
Admin functions are mostly concerned with creating, showing, updating, and deleting data (dashboard) same as before but instead of responding with json, the response will be rendering a view (blade page).

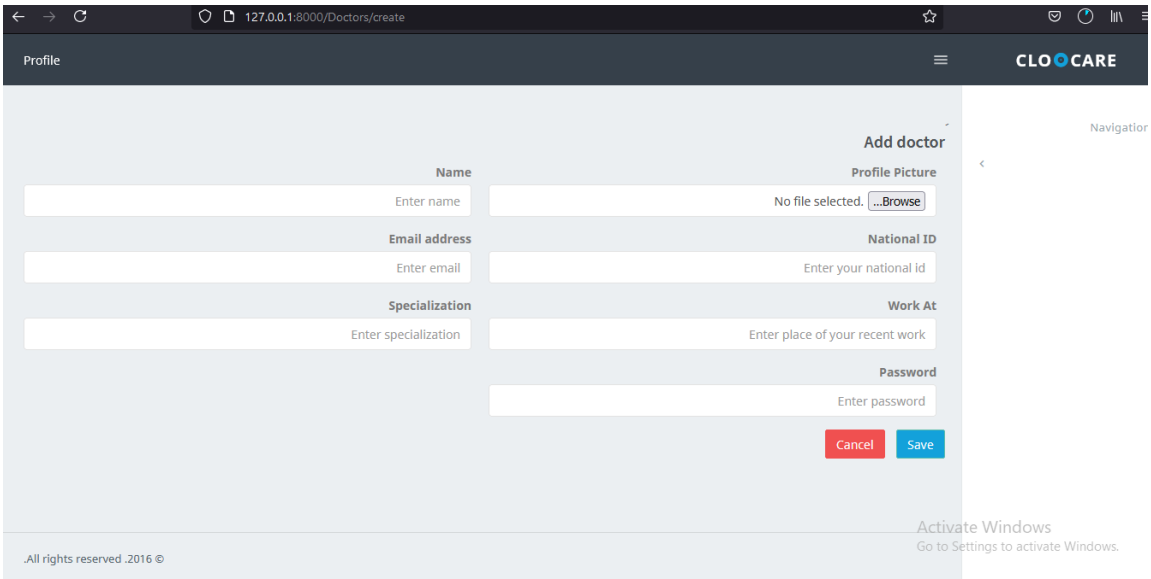


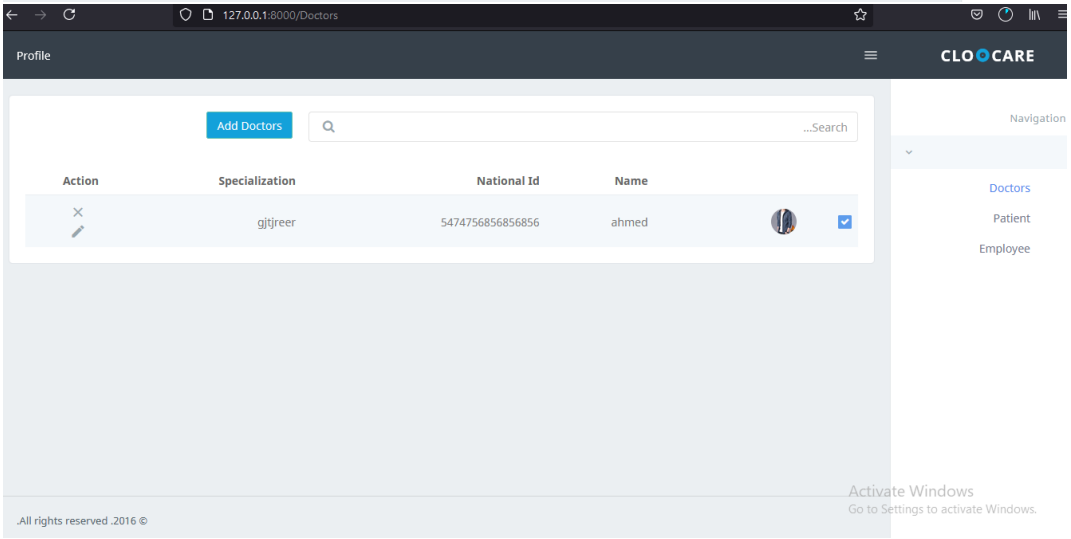

127.0.0.1:8000/Patients/1
Profile
CLOCARE
Patient "amera" Details
Navigation
amera
6676123467856789 National Id
8574596945 Phone number
fdfhgfgj Address
2021-06-23 Birth date
2021-06-15 Arriving at
77 Age
2021-06-23 Birth date
88 height
66 weight
Diseases1
Diseases2
Activate Windows
Go to Settings to activate Windows.

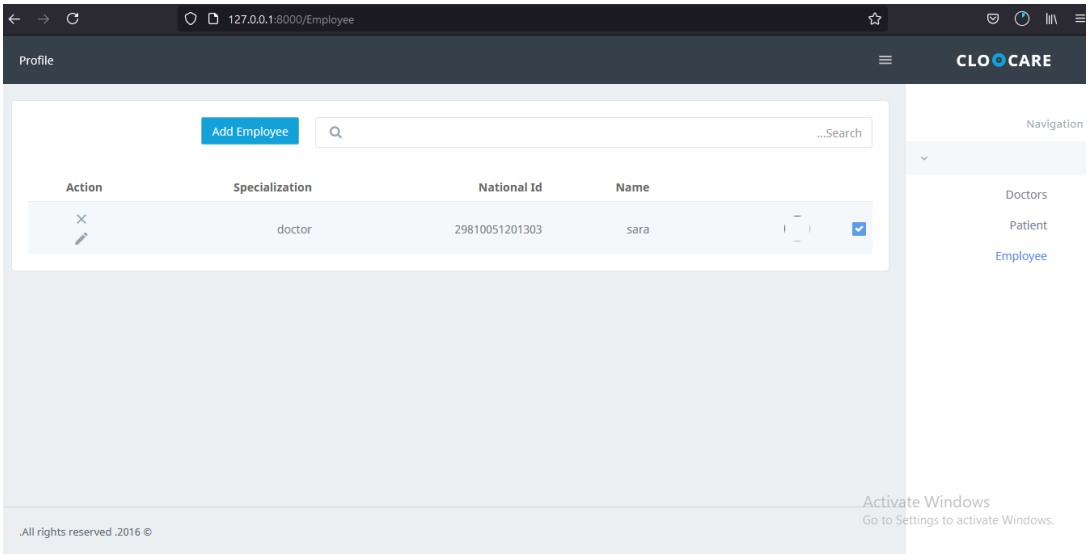

127.0.0.1:8000/Patients/1/edit
Profile
CLOCARE
Navigation
profile photo
No f...ed.
...Browse
National ID
6676123467856789
Name
amera
Address
fdfhgfgj
Birth date
06 / 23 / 2021
Phone number
8574596945
arriving_data
06 / 15 / 2021
male
gender
hospital name
dhdgfjh
Age
77
Weight
66
Height
88
Blood type
+a
Medicines
Enter a medicine
dfgfdgdf
asfsdf
Enter a medicine
Enter a medicine
Enter a medicine
Cancel
Update
Activate Windows

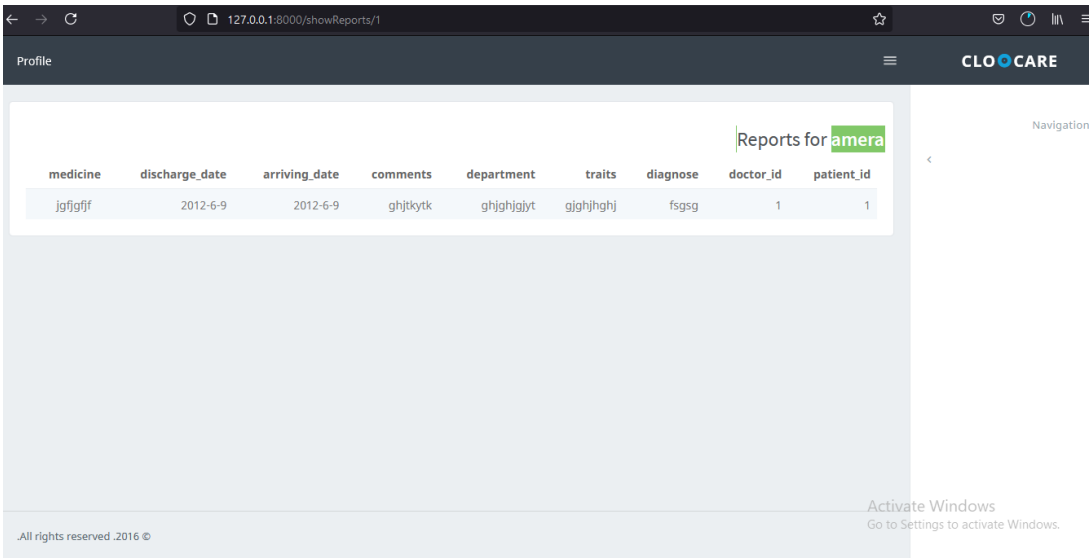

127.0.0.1:8000/Patients
Profile
CLOCARE
In patients "0"
out patients "2"
Add patient+
...Search
Navigation
Action
other controllers
Phone number
National Id
Name
reports
sensors
_tests
8574596945
6676123467856789
amera
reports
sensors
_tests
56587868
67876978978989
sarra
Doctors
Patient
Employee
.All rights reserved .2016 ©
Activate Windows
Go to Settings to activate Windows.

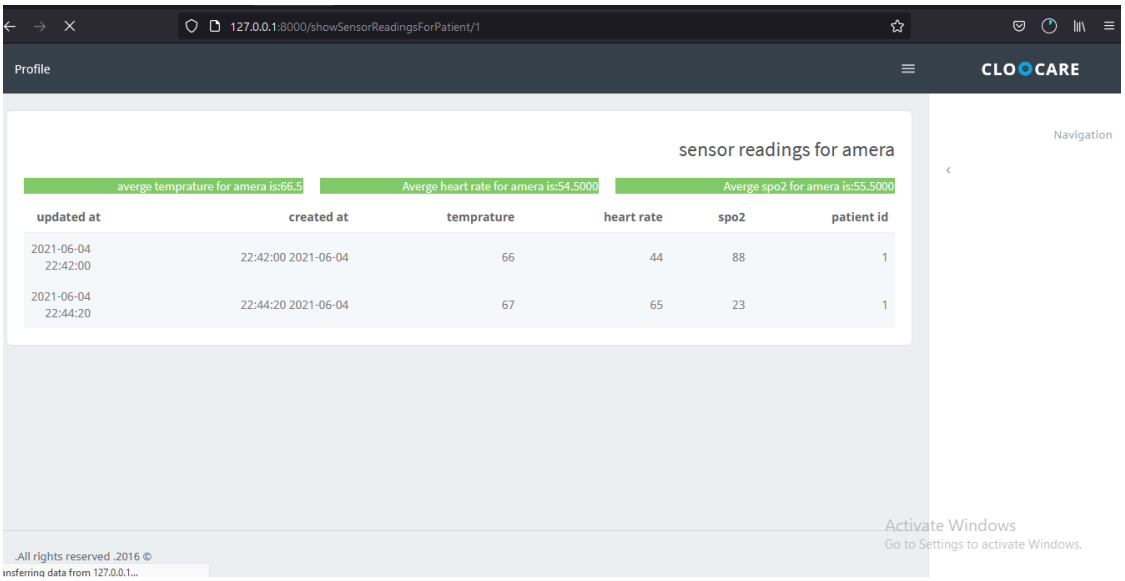

127.0.0.1:8000/Doctors/1/edit
Profile
CLOCARE
Navigation
Upload a different photo
No file selected. ...Browse
National ID
5474756856856856
Name
ahmed
Work At
drgdrdfbd
Email address
sfsfsdf@gdfg.dsgsg
Specialization
gjtjreer
Cancel
Update
Activate Windows
Go to Settings to activate Windows.
.All rights reserved .2016 ©

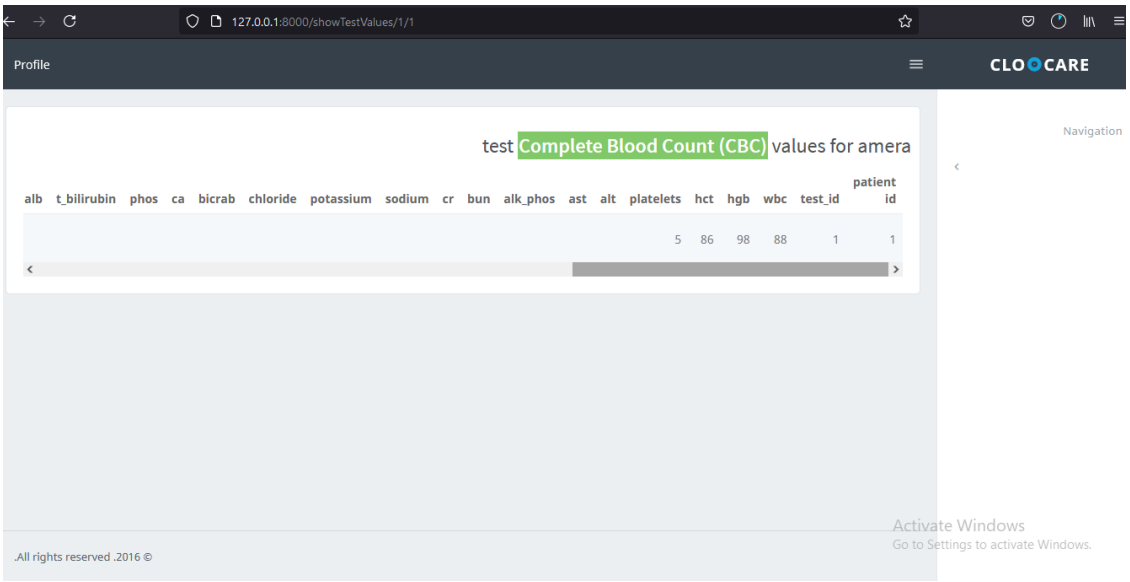

127.0.0.1:8000/Doctors/create
Profile
CLOCARE
Navigatior
Add doctor
Name
Enter name
Profile Picture
No file selected. ...Browse
Email address
Enter email
National ID
Enter your national id
Specialization
Enter specialization
Work At
Enter place of your recent work
Password
Enter password
Cancel
Save
Activate Windows
Go to Settings to activate Windows.
.All rights reserved .2016 ©

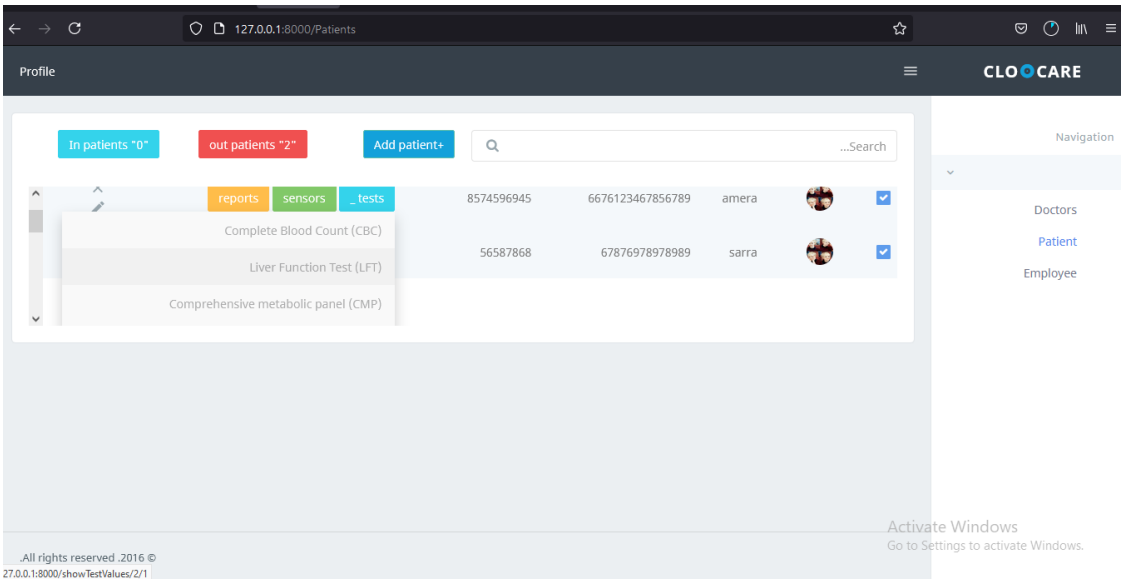

127.0.0.1:8000/Doctors
Profile
CLOCARE
Add Doctors
...Search
Navigation
Action
Specialization
National Id
Name
gjtjreer
5474756856856856
ahmed
Doctors
Patient
Employee
Activate Windows
Go to Settings to activate Windows.
.All rights reserved .2016 ©


127.0.0.1:8000/Employee
Profile
CLOCARE
Add Employee
...Search
Action
Specialization
National Id
Name
doctor
29810051201303
sara
Navigation
Doctors
Patient
Employee
.All rights reserved .2016 ©
Activate Windows
Go to Settings to activate Windows.


127.0.0.1:8000/showReports/1
Profile
CLOCARE
Reports for amera
medicine
discharge_date
arriving_date
comments
department
traits
diagnose
doctor_id
patient_id
jgfjgfjf
2012-6-9
2012-6-9
ghjtkytk
ghjghjgjyt
gjghjhghj
fsgsg
1
1
Navigation
.All rights reserved .2016 ©
Activate Windows
Go to Settings to activate Windows.


127.0.0.1:8000/showSensorReadingsForPatient/1
Profile
CLOCARE
sensor readings for amera
averge temprature for amera is:66.5
Averge heart rate for amera is:54.5000
Averge spo2 for amera is:55.5000
updated at
created at
temprature
heart rate
spo2
patient id
2021-06-04 22:42:00
22:42:00 2021-06-04
66
44
88
1
2021-06-04 22:44:20
22:44:20 2021-06-04
67
65
23
1
Navigation
.All rights reserved .2016 ©
Activate Windows
Go to Settings to activate Windows.


127.0.0.1:8000/showTestValues/1/1
Profile
CLOCARE
Navigation
test Complete Blood Count (CBC) values for amera
alb t_bilirubin phos ca bicrab chloride potassium sodium cr bun alk_phos ast alt platelets hct hgb wbc test_id patient id
5 86 98 88 1 1
.All rights reserved .2016 ©
Activate Windows
Go to Settings to activate Windows.

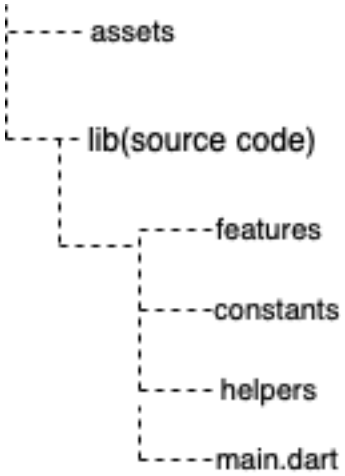

127.0.0.1:8000/Patients
Profile
CLOCARE
Navigation
In patients "0"
out patients "2"
Add patient+
...Search
reports
sensors
_ tests
8574596945
6676123467856789
amera
Complete Blood Count (CBC)
Liver Function Test (LFT)
Comprehensive metabolic panel (CMP)
56587868
67876978978989
sarra
Doctors
Patient
Employee
.All rights reserved .2016 ©
27.0.0.1:8000/showTestValues/2/1
Activate Windows
Go to Settings to activate Windows.

3.2 Mobile Application

3.2.1 Project Structure & Architecture:

3.2.1.1 Folder Structure

The mobile app project is structured as seen in the figure below:

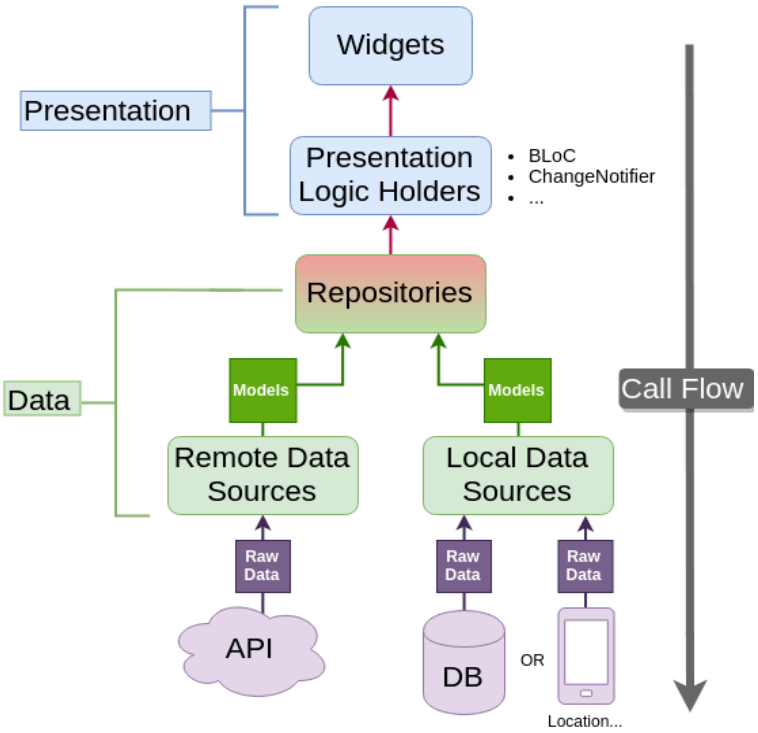


From the previous figure the project contains two main folders

- Asset folder: which contains all the images used in the app
- Lib folder: And this one contains the source code of the app and it consists of the following:
    - Main.dart: The starting point of the app
    - Helpers: Reusable widgets & classes that are used throughout the app are defined here
    - Constants: Contains static styles like fonts and colors that are used in the app, also contains auto generated strings to easily call the available resources in the assets folder
    - Features: This one contains all the features that the app consists of (Authentication, Home, Profile ...etc.) will be discussed in detail in the coming chapter

3.2.1.2 Architecture Design

- The app adopts a simplified version of Clean Architecture proposed by Robert Martin as seen in the figure below:



- The app is collection of features each feature as seen in the figure above consists of the following:
  1) Data layer: This layer is responsible for handling and parsing any data that's coming from outside the app, e.g.: REST responses and local storage date, it has two sub-layers
     a) Repository: Fetching or sending data is done in this sub-layer
     b) Models: fetched data is parsed into dart objects through model. The following piece of code is a working example of the data layer:

```
Future<AuthModel> login(String name, String password) async {
  final url = AppUrls.login;
  Map<String, dynamic> body = {
    "national_id": "$name",
    "password": "$password"
  };
  return AuthModel.fromJson(await client.postRequest(url, body));
}
```

This function as the name suggests, logins the user by sending his credentials to the defined Url and fetches the response in Map form The response is the parsed through the authentication model into a dart object that can be easily dealt with.

2) Presentation layer: State management and UI rendering are done in this layer, it also consists of two sub-layers:
   a) State: Business LOgic Component (BLoC) this sub-layer manages the state of the app based on the data coming from the Data layer and renders the UI based on the current state either it's loading, has data or has error, that's done using a state management called Cubit which is the concept as BLoC but in a compact form. The figure below shows a working example of how Cubit works

```
login() async {
  emit(BaseState.onLoading());
  try {
    model = repo.login(nameCtrler.text, passwordCtrler.text);
    emit(BaseState.onData(model));
  } catch (e) {
    print('error login: $e');
    emit(BaseState.onError('$e'));
  }
}
```

After we fetched and parsed the login response coming from the repository, we manipulate the state in the auth cubit so we can update the UI based on the current state,
First we set the current state to loading by calling emit and passing the loading factory BaseState.onLoading
Then we call the login function from our AuthRepo and wait until it finishes response with either a data response or error response each with the corresponding state factory onData, onError respectively.

b) UI: contains all widgets making up the screen based on the current state And finally, we come that last part of the string, the UI will be listening to every state change that occurs in the state class and rendered the UI based on the current state as seen in the following code snippet:

```
return BlocConsumer<AuthBloc, BaseState>(
  bloc: bloc,
  listener: (context, state) {
    if (state is ListenerHasData)
      DoctorHomeScreen().pushAndRemoveAll(context);
    if (state is ListenerHasError)
      showFlushBar(
      context, 'Login failed');
    },
  builder: (context, state) {
    if (state is ListenerLoading)
      return CircularProgressIndicator();
    return _buildBtn();
  },
);
```

Here we have a bloc object instantiated from the previously mentioned AuthCubit
this bloc object will be listened to by the CubitBuilder widgets, so that when a loading state is emitted from the bloc object on pressing login for example a progress indicator will be rendered, after a short while a new state will be emitted
in case it ended with an error a SnackBar will be shown with the returned error and the same login will be rendered so that the user can retry to login, on the the hand
if it ended with a data state meaning that the user has successfully logged in he will be redirected to the homeScreen and perform the available actions as he wishes

3.2.2 Tools and packages

3.2.2.1 Languages and Framework

- Dart language: A language developed by Google and is language Flutter Framework is based on
- Flutter framework: An open-source multi-platform framework based on Dart language, developed and maintained by google and the community, chosen for its simplicity and reliability as it's maintained by the same company that maintains the Android platform which is a huge plus as this will allow for more dependability, Flutter also allows for releasing both Android & iOS with the same codebase and performance so close to native performance.

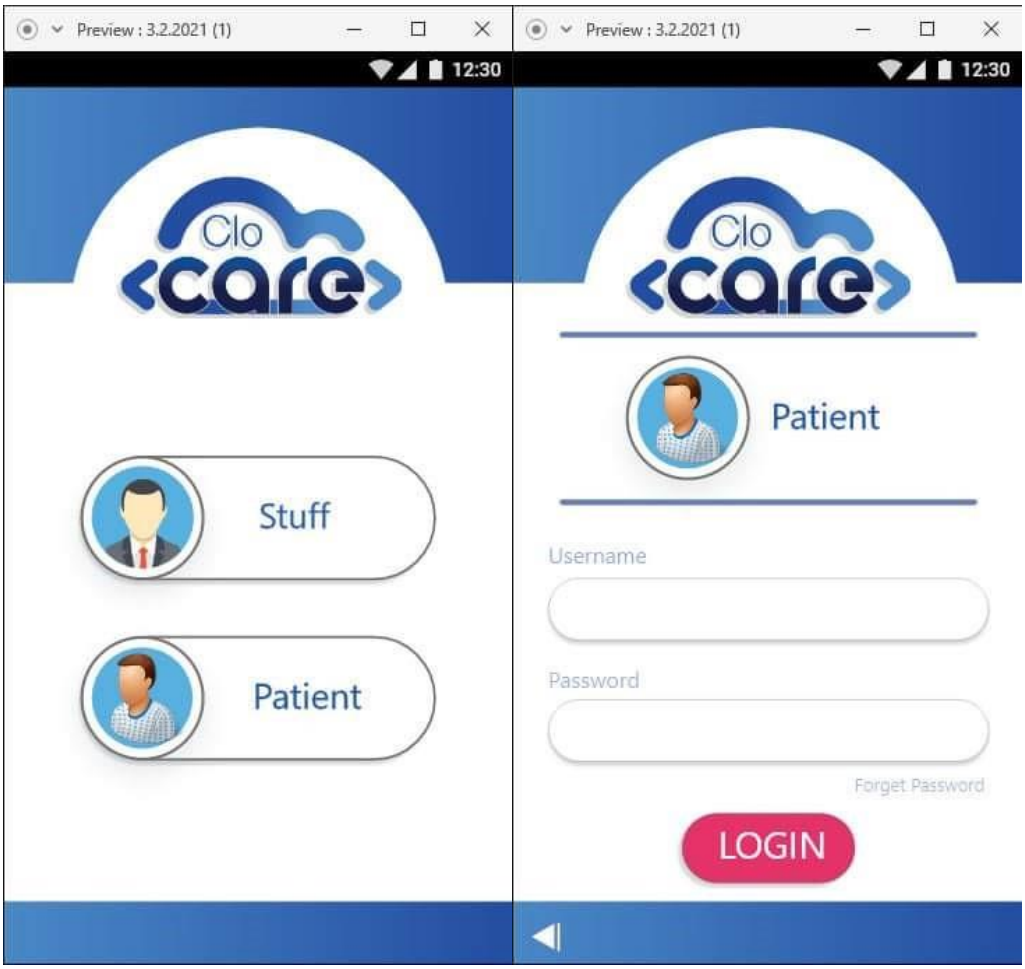
3.2.2.2 Tools used for making the project

- VSCode: Code editor developed by Microsoft, chosen for this project for its good support for Flutter & Dart plugins and for being lightweight and convenient to use
- Flutter & Dart Plugins: Plugins installed through VSCode store to support Flutter framework and Dart language, they facilitate the development process by allowing code competition and shortcuts
- Flutter SDK: This SDK is needed to develop apps using flutter

3.2.3 Features

3.2.3.1 Authentication

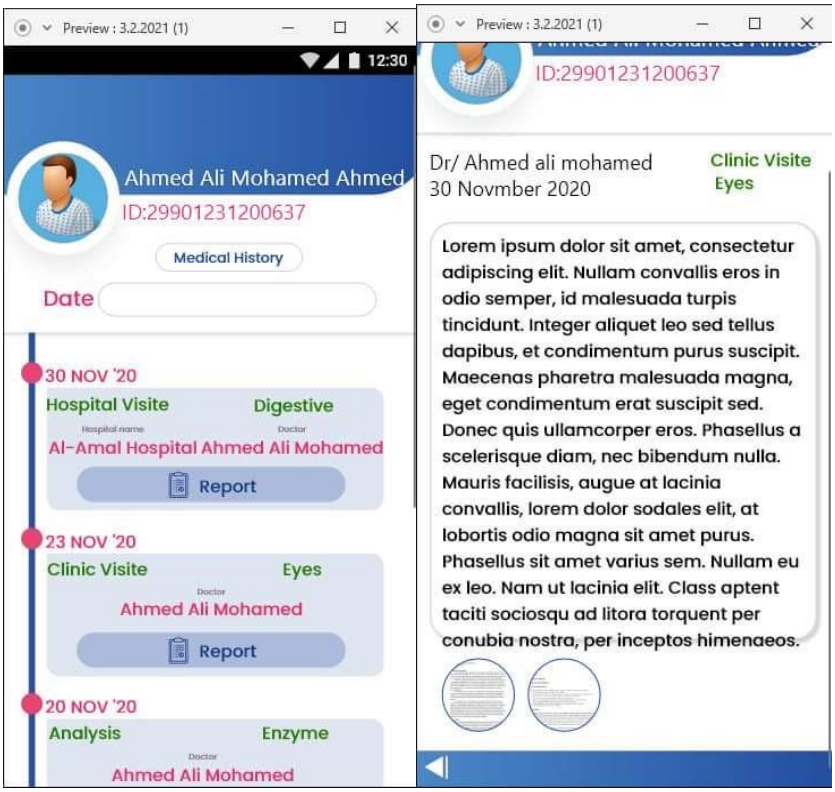
- When the app is launched the user is first presented with the role selection screen as seen in figure 1 asking them to choose a role whether he wishes to log in as a User or Doctor each role will be discussed in details later on
- After choosing a role the user is sent to the login page as in figure 2 based on his selected role



3.2.3.2 Patient Mod

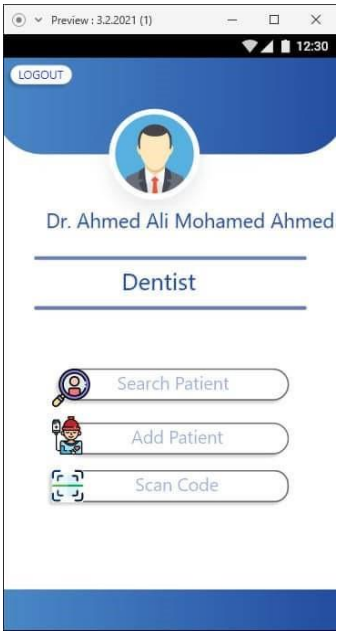
- If the user entered a valid patient Id number and password he will be redirected to his profile where he's only allowed to view his profile details as seen in Figure 3
- The user can also view the reports doctors made about him as seen in Figure 4, 5

Preview : 3.2.2021 (1)
12:30
LOGOUT
Ahmed Ali Mohamed Ahmed
ID:29901231200637
01002458867
30
23/01/1990
A+
5'7"ft
Height
135
Weight
Egypt - Eladkahlya - Elmansoura
Diseases

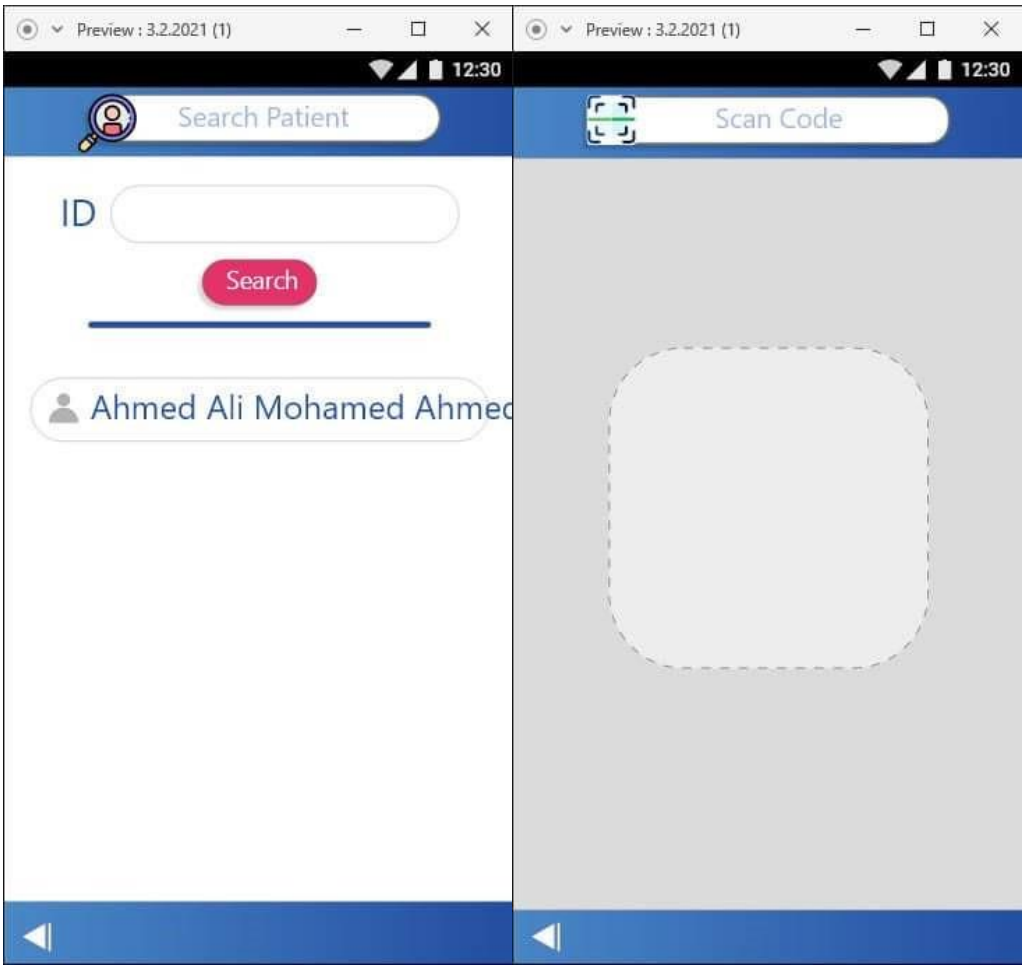

Preview : 3.2.2021 (1)
12:30
Ahmed Ali Mohamed Ahmed
ID:29901231200637
Medical History
Date
30 NOV '20
Hospital Visite
Digestive
Al-Amal Hospital
Ahmed Ali Mohamed
Report
23 NOV '20
Clinic Visite
Eyes
Ahmed Ali Mohamed
Report
20 NOV '20
Analysis
Enzyme
Ahmed Ali Mohamed
Preview : 3.2.2021 (1)
ID:29901231200637
Dr/ Ahmed ali mohamed
30 Novmber 2020
Clinic Visite
Eyes
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam convallis eros in odio semper, id malesuada turpis tincidunt. Integer aliquet leo sed tellus dapibus, et condimentum purus suscipit. Maecenas pharetra malesuada magna, eget condimentum erat suscipit sed. Donec quis ullamcorper eros. Phasellus a scelerisque diam, nec bibendum nulla. Mauris facilisis, augue at lacinia convallis, lorem dolor sodales elit, at lobortis odio magna sit amet purus. Phasellus sit amet varius sem. Nullam eu ex leo. Nam ut lacinia elit. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.

3.2.3.3 Doctor Mode

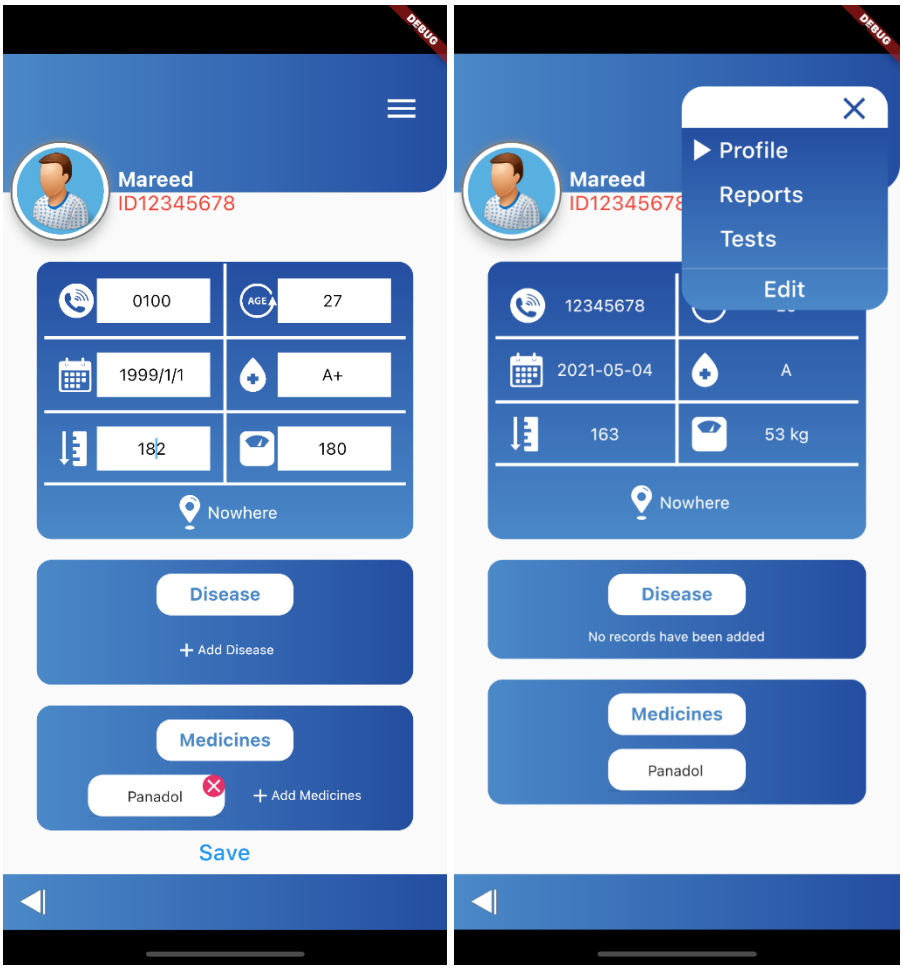
- On the other hand, if the chosen role is Doctor and a valid account credentials were entered, the user will instead be redirected to the home screen where has more to do that to just view as seen in Figure 6
- Here we have two main functions
    - Search Patient
    - Scan patient
- Both functions aim to find the id of the patient under examination so that we can view edit or add information to their profile



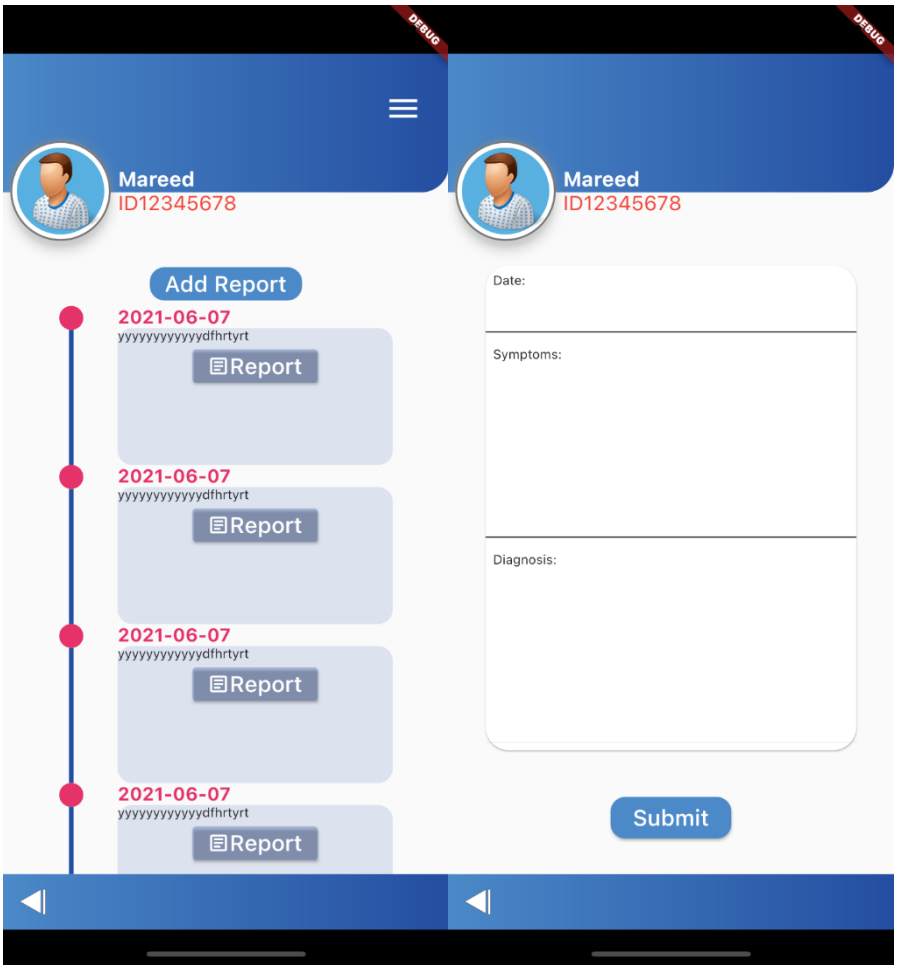
- Search patient: if search patient is chosen, the doctor will be redirected to the page as seen in figure 7 where he can start entering the patient's id to get instant results of the available patients starting with the entered id
- Scan patient: patient is found here through a RFID tag with their id written on it that can be scanned using any NFC enabled mobile phone as seen in Figure in 8



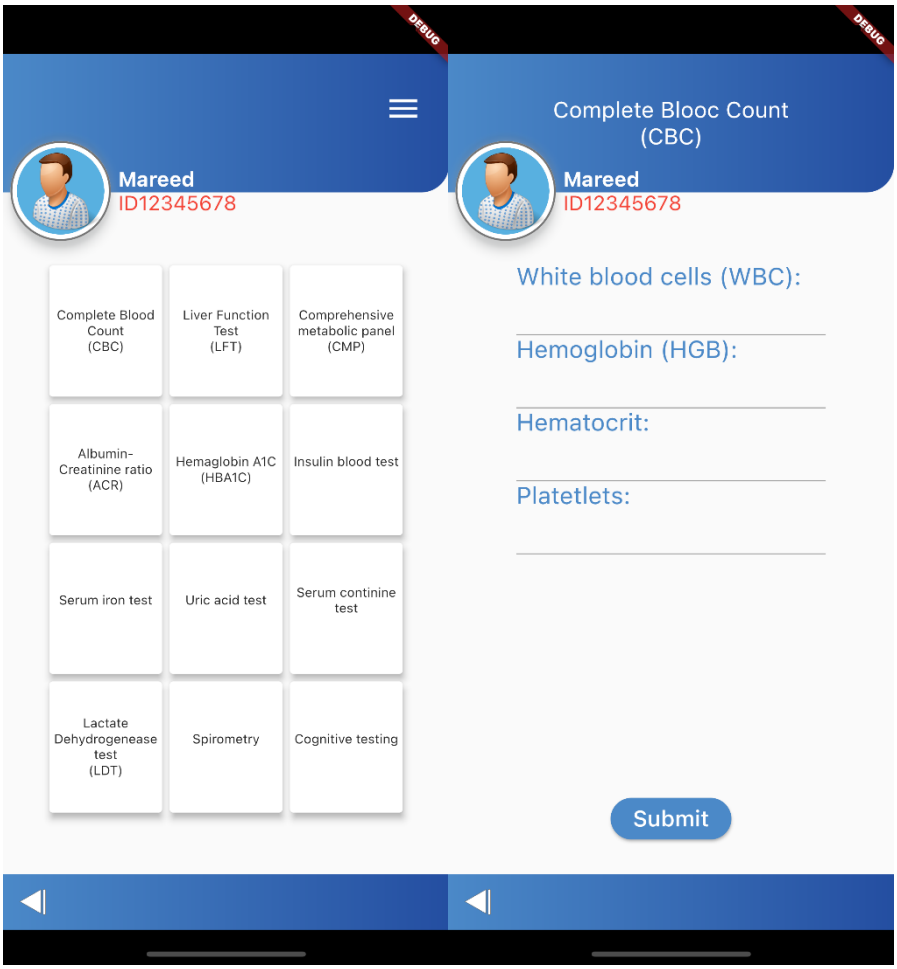
- After choosing the patient of interest either by searching his id or by scanning his RFID tag, the doctor will be redirected to the patient's profile which is the same as the one appeared to the patient in Figure 3 but with more actions which can be accessed through the options menu in the top right corner of the screen



- From the options menu doctors can view reports that are related to the patient of interest (Figure 10) or add new ones (Figure 11) that can be later viewed by other users (doctors or patients) when viewing this patient's profile



- Third action we have in the options menu is Tests here we can add and view tests that have been performed on the patients as seen in figure 12, Here we have a list of all the available tests that the doctor can perform
  When a particular test has been chosen, the user will be redirected to the test details screen as seen in figure 13, where all the attributes related to the selected test will be present as text fields that the doctor can fill then submit the test to add to the patient tests that can be viewed later on



- Last action we have is Edit as the name suggests, on pressed the doctor will be able to edit the patient's profile values as seen in Figure 13, a doctor can edit the patient's weight, height, add or remove a disease or medicine ...etc.

3.3 Front End

3.3.1 Introduction

3.3.1.1 What is Front-End?

Simply the front end is anything appear for you as a user when you open any web page.

3.3.1.2 What you basically need to make front-end web page?

To make a front-end web page you need to use HTML, CSS, JavaScript (JS). You also need a design to cope with the target of your website and to follow it while coding your webpages.

3.3.1.3 Front-end focused

- Markup and web languages such as HTML, CSS, JavaScript, and ancillary libraries commonly used in those languages such as Sass or jQuery.
- Asynchronous request handling and AJAX
- Single-page applications (with frameworks like React, AngularJS or Vue.js)
- Web performance (largest contentfull paint, time to interactive, 60 FPS animations and interactions, memory usage, etc.)
- Responsive web design
- Cross-browser compatibility issues and workarounds
- End-to-end testing with a headless browser
- Build automation to transform and bundle JavaScript files, reduce images size... with tools like Webpack or Gulp.js.
- Search engine optimization, Basic usage of image editing tools such as GIMP or Photoshop, User Interface.

3.3.2 ALL Used Tools

- You even can’t imagine becoming a key player of Front-End Development by skipping or ignoring these three technologies – HTML, CSS & JavaScript. These are the building blocks of front-end development. All in all, HTML (HyperText Markup Language) is concerned with the structure of the web page whereas CSS (Cascading Style Sheets) looks after the styling of the web page. Moreover, if we talk about JavaScript, it is the client-side programming language that is used to implement interactive functionalities such as animations, audio/videos, maps, and many more on the website.
- You need to learn the syntax, features & implementation of these languages for creating a website. There are various online platforms available over the

web from where you can learn these technologies and can practice your skills as well.

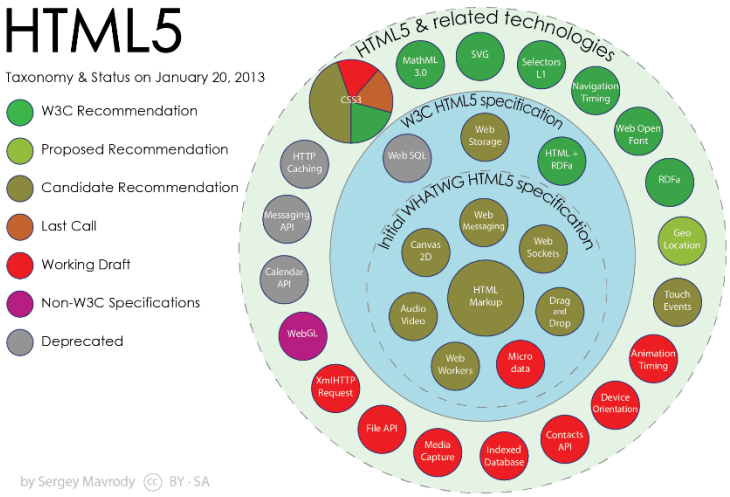
- Once you'll get done with the HTML, CSS & JavaScript, now you need to learn several Front-End frameworks as per your requirement and convenience. In general, the Front-End frameworks are libraries of already written codes and these frameworks make the front-end development easier and efficient due to various features such as reusable components, etc. If we particularly talk about CSS, Bootstrap is the most popular framework in this domain. Moreover, there are various other renowned frameworks such as Angular, React, Vue, etc. that can be taken into consideration for frontend development.

3.3.2.1 HTML

HTML is the standard markup language for creating Web pages.

- What is HTML?
  - HTML stands for Hyper Text Markup Language.
  - HTML is the standard markup language for creating Web pages.
  - HTML describes the structure of a Web page.
  - HTML consists of a series of elements.
  - HTML elements tell the browser how to display the content.
  - HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

- HTML (HyperText Markup Language) is the most basic building block of the Web. It defines the meaning and structure of web content. Other technologies besides HTML are generally used to describe a web page's appearance/presentation (CSS) or functionality/behavior (JavaScript).
- "Hypertext" refers to links that connect web pages to one another, either within a single website or between websites. Links are a fundamental aspect of the Web. By uploading content to the Internet and linking it to pages created by other people, you become an active participant in the World Wide Web.
- HTML uses "markup" to annotate text, images, and other content for display in a Web browser. HTML markup includes special "elements" such as <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, , , <img>, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <output>, <progress>, <video>, <ul>, <ol>, <li> and many others.
- An HTML element is set off from other text in a document by "tags", which consist of the element name surrounded by "<" and ">". The name of an element inside a tag is case insensitive. That is, it can be written in uppercase, lowercase, or a mixture. For example, the <title> tag can be written as <Title>, <TITLE>, or in any other way.
- Some used properties:
    - HTML media
    - HTML graphic
    - HTML forms
    - HTML APIS

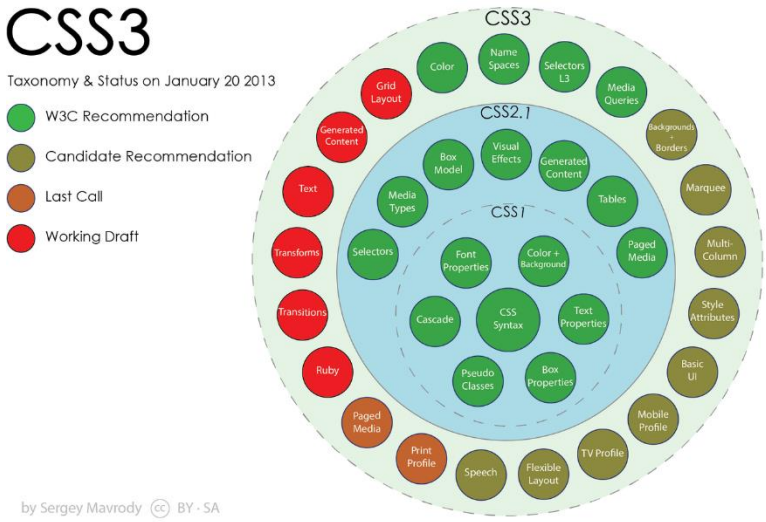


3.3.2.2 CSS

CSS is the language we use to style a Web page.

- What is CSS?
  - CSS stands for Cascading Style Sheets
  - CSS describes how HTML elements are to be displayed on screen, paper, or in other media.
  - CSS saves a lot of work. It can control the layout of multiple web pages all at once.
  - External stylesheets are stored in CSS files.
- Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.[1] CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.
- CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts.
- This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate CSS file which reduces complexity and repetition in the structural content as well as enabling the CSS file to be cached to improve the page load speed between the pages that share the file and its formatting.
- Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.
- The name cascading comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element. This cascading priority scheme is predictable.
- The CSS specifications are maintained by the World Wide Web Consortium (W3C). Internet media type (MIME type) text/css is registered for use with CSS by RFC 2318 (March 1998). The W3C operates a free CSS validation service for CSS documents.
- Some used properties:
  - Gradients
  - Transforms
  - Transitions
  - Animations

- Typewriter animation
- Responsive CSS
- RWD media queries



3.3.2.3 JavaScript

JavaScript is the world's most popular programming language. JavaScript is the programming language of the Web. JavaScript is easy to learn. JavaScript often abbreviated as JS, is a programming language that conforms to the ECMAScript specification.

JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.

Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. Over 97% of websites use it client-side for web page behavior, often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute the code on the user's device.

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

The ECMAScript standard does not include any input/output (I/O), such as networking, storage, or graphics facilities. In practice, the web browser or other runtime system provides JavaScript APIs for I/O.

JavaScript engines were originally used only in web browsers, but they are now core components of other software systems, most notably servers and a variety of applications.

3.3.2.4 JQuery

jQuery is a JavaScript Library. jQuery greatly simplifies JavaScript programming. The purpose of jQuery is to make it much easier to use JavaScript on your website.

- What Should You Already Know?
  - Before you start studying jQuery, you should have a basic knowledge of:
    - HTML
    - CSS
    - JavaScript
- What is jQuery?
  - jQuery is a lightweight, "write less, do more", JavaScript library.
  - The purpose of jQuery is to make it much easier to use JavaScript on your website.
  - jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish and wraps them into methods that you can call with a single line of code.
  - jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.
- Why jQuery?
  - There are lots of other JavaScript libraries out there, but jQuery is probably the most popular, and also the most extendable.
  - Many of the biggest companies on the Web use jQuery, such as:
    - Google
    - Microsoft
    - IBM
    - Netflix
- Does jQuery work in all browsers?
  - The jQuery team knows all about cross-browser issues, and they have written this knowledge into the jQuery library. jQuery will run the same in all major browsers.

3.3.2.5 Bootstrap

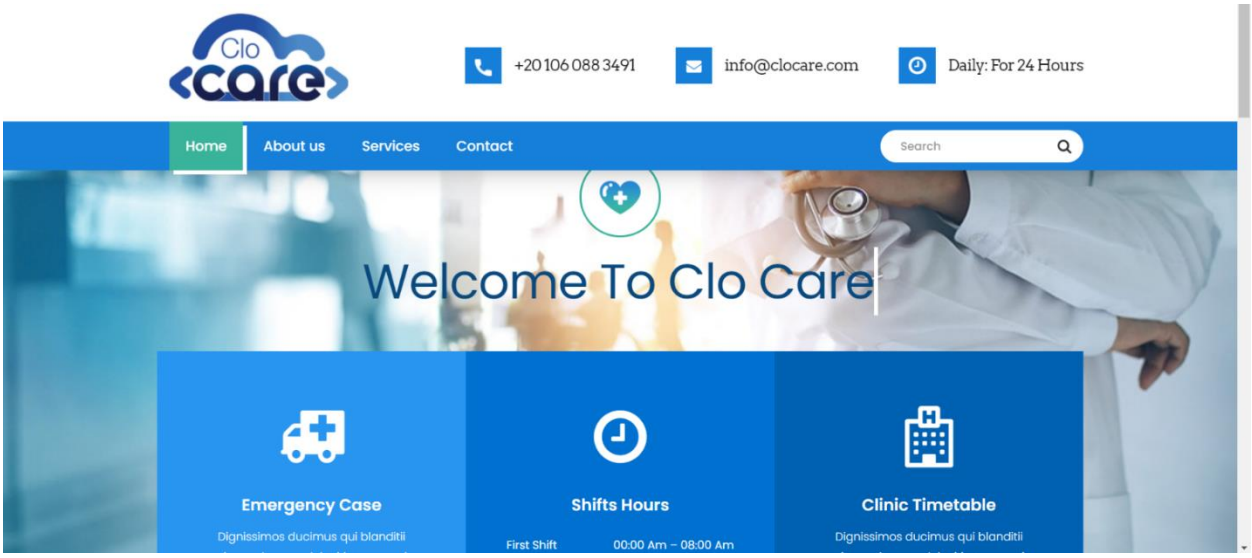
Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites. Bootstrap is completely free to download and use!

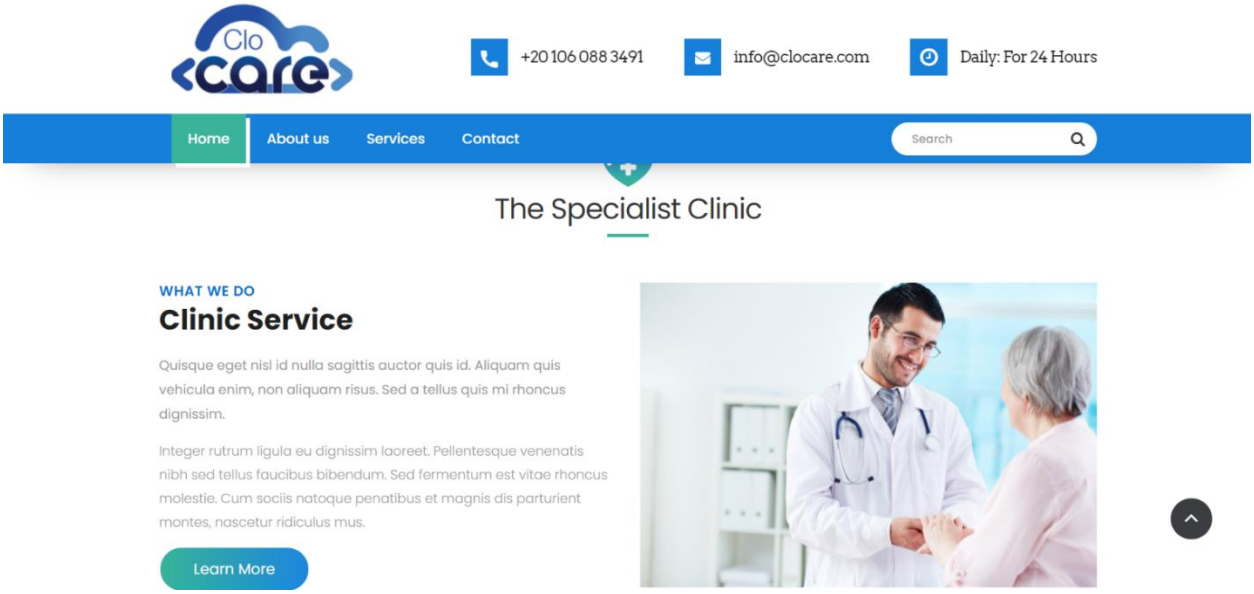
- What is Bootstrap?
  - Bootstrap is a free front-end framework for faster and easier web development.

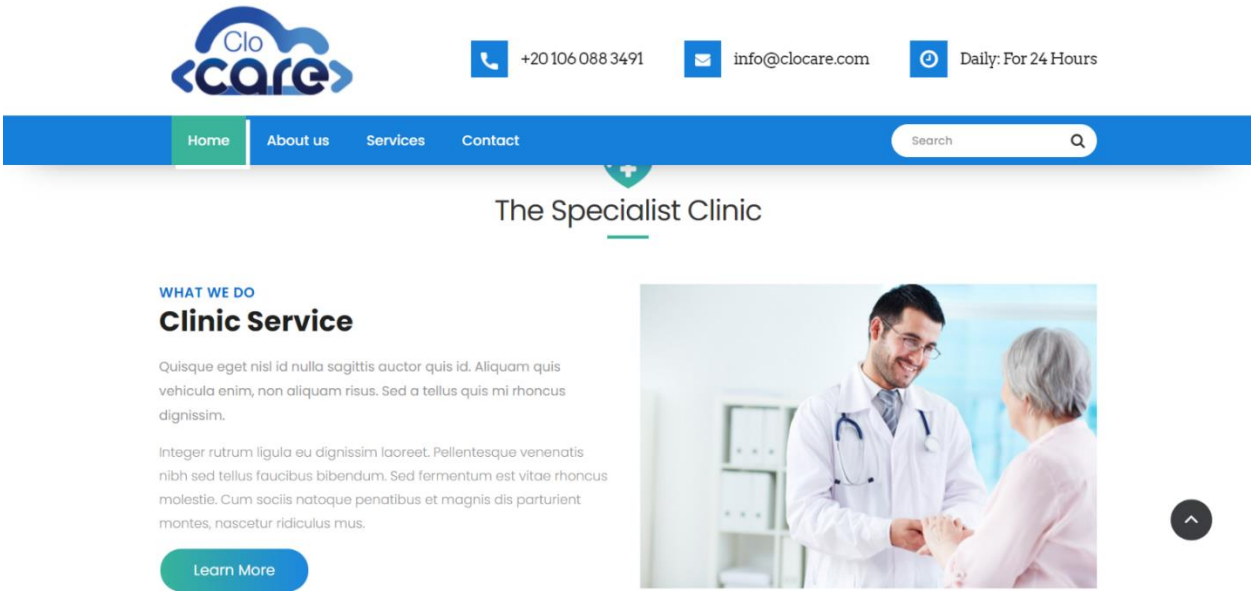
- Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other, as well as optional JavaScript plugins.
- Bootstrap also gives you the ability to easily create responsive designs.

- What is Responsive Web Design?
  - Responsive web design is about creating web sites which automatically adjust themselves to look good on all devices, from small phones to large desktops.
- Why Use Bootstrap?
  - Advantages of Bootstrap:
    - Easy to use: Anybody with just basic knowledge of HTML and CSS can start using Bootstrap.
    - Responsive features: Bootstrap's responsive CSS adjusts to phones, tablets, and desktops.
    - Mobile-first approach: In Bootstrap 3, mobile-first styles are part of the core framework.
    - Browser compatibility: Bootstrap is compatible with all modern browsers (Chrome, Firefox, Internet Explorer, Edge, Safari, and Opera).
- Where to Get Bootstrap?
  - There are two ways to start using Bootstrap on your own web site.
  - You can:
    - Download Bootstrap from getbootstrap.com.
    - Include Bootstrap from a CDN.

3.3.2.6 Front-End Design & Coding




Clo
care
+20 106 088 3491
info@clocare.com
Daily: For 24 Hours
Home
About us
Services
Contact
Search
The Specialist Clinic
WHAT WE DO
Clinic Service
Quisque eget nisl id nulla sagittis auctor quis id. Aliquam quis vehicula enim, non aliquam risus. Sed a tellus quis mi rhoncus dignissim.
Integer rutrum ligula eu dignissim laoreet. Pellentesque venenatis nibh sed tellus faucibus bibendum. Sed fermentum est vitae rhoncus molestie. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.
Learn More


Clo
care
+20 106 088 3491
info@clocare.com
Daily: For 24 Hours
Home
About us
Services
Contact
Search
The Specialist Clinic
WHAT WE DO
Clinic Service
Quisque eget nisl id nulla sagittis auctor quis id. Aliquam quis vehicula enim, non aliquam risus. Sed a tellus quis mi rhoncus dignissim.
Integer rutrum ligula eu dignissim laoreet. Pellentesque venenatis nibh sed tellus faucibus bibendum. Sed fermentum est vitae rhoncus molestie. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.
Learn More

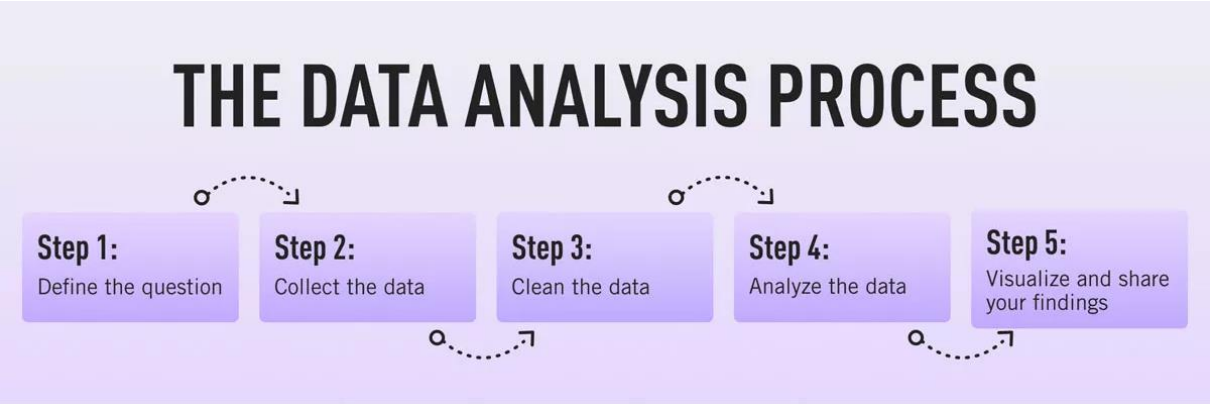
Chapter 4

Data Analysis

4.1 Data Analysis Process

4.1.1 Introduction

Nowadays, data has become abundantly available. Almost all institutions have a way to store their data and the data of all work and operations related to it. Government institutions, companies of all kinds, banks, and most organizations need to take decisions that affect their work, their future, or the lives of other people. Hence the need to investigate the available data on its workflow or the nature of the environment around it in order to take the necessary decisions. As for government institutions, making a wrong decision would affect the lives of citizens in all directions and affect the future of politicians, perhaps the economy, investment, or foreign relations with other countries. On the other hand, the world of business and finance is highly volatile and difficult to anticipate. Therefore, there is a need for a reliable means to anticipate this field and take appropriate decisions. The science of data analysis appears to be the solution and to make it easier for decision makers and businessmen to make their decisions through a comprehensive process conducted on specific data.



Data analysis is the process of collecting, cleaning, exploring, and analyzing data in order to answer specific questions and discover the required information needed about the data. The results obtained are communicated to the meant people, suggesting conclusions, and supporting decision-making. The procedure helps reduce the risks inherent in decision-making by providing useful insights and statistics, often presented in charts, images, tables, and graphs.

There are five popular types of data analysis available today, commonly employed in the worlds of technology and business. They are:

- **Diagnostic Analysis:** Diagnostic analysis answers the question, “Why did this happen?” Using insights gained from statistical analysis, analysts use diagnostic analysis to identify patterns in data. Ideally, the analysts find similar patterns that

existed in the past, and consequently, use those solutions to resolve the present challenges hopefully.

- **Predictive Analysis:** Predictive analysis answers the question, "What is most likely to happen?" By using patterns found in older data as well as current events, analysts predict future events. While there's no such thing as 100 percent accurate forecasting, the odds improve if the analysts have plenty of detailed information and the discipline to research it thoroughly.
- **Statistical Analysis:** Statistical analysis answers the question, "What happened?" This analysis covers data collection, analysis, modeling, interpretation, and presentation using dashboards. The statistical analysis breaks down into two sub-categories:
    - **Descriptive:** Descriptive analysis works with either complete or selections of summarized numerical data. It illustrates means and deviations in continuous data and percentages and frequencies in categorical data.
    - **Inferential:** Inferential analysis works with samples derived from complete data. An analyst can arrive at different conclusions from the same comprehensive data set just by choosing different samplings.
- **Text Analysis:** Also called "data mining," text analysis uses databases and data mining tools to discover patterns residing in large datasets. It transforms raw data into useful business information. Text analysis is arguably the most straightforward and the most direct method of data analysis.

4.1.2 Posing questions

The first step in any data analysis process is to define the objective. In data analytics jargon, this is sometimes called the 'problem statement'. Defining the objective means coming up with a hypothesis and figuring how to test it. Start by asking: why am I doing this analysis? What business problem am I trying to solve? or why are we losing customers?

While this might sound straightforward, it can be trickier than it seems. A data analyst's job is to understand the business and its goals in enough depth that they can frame the problem the right way.

The data required for analysis is based on a question or an experiment. Based on the requirements of those directing the analysis, the data necessary as inputs to the analysis is identified (e.g., Population of people). Specific variables regarding a population (e.g., Age and Income) may be specified and obtained. Data may be numerical or categorical.

4.1.3 Data Collection

Data Collection is the process of gathering information on targeted variables identified as data requirements. The emphasis is on ensuring accurate and honest collection of data. Data Collection ensures that data gathered is accurate such that the related decisions are valid. Data Collection provides both a baseline to measure and a target to improve.

Data is collected from various sources ranging from organizational databases to the information in web pages. The data thus obtained, may not be structured, and may contain irrelevant information. Hence, the collected data is required to be subjected to Data Processing and Data Wrangling.

4.1.4 Data wrangling

Data Wrangling is the process of preventing and correcting the errors in the dataset. Once collecting the needed data, the next step is to get it ready for analysis. This means cleaning, or 'scrubbing' it, and this is crucial in making sure that you are working with high-quality data. There are several types of Data Cleaning that depend on the type of data. For example, while cleaning the financial data, certain totals might be compared against reliable published numbers or defined thresholds. Likewise, quantitative data methods can be used for outlier detection that would be subsequently excluded in analysis.

Key data cleaning tasks include:

- **Removing major errors, duplicates, and outliers**—all of which are inevitable problems when aggregating data from numerous sources.
- **Removing unwanted data points**—extracting irrelevant observations that have no effect on your intended analysis.
- **Bringing structure to your data**—general 'housekeeping', i.e., fixing typos or layout issues, which will help you map and manipulate your data more easily.
- **Filling in major gaps**—as you are tidying up, you might notice that important data are missing. Once you have identified gaps, you can go about filling them.

4.1.5 Exploratory data analysis

Another thing many data analysts do (alongside cleaning data) is to carry out an exploratory analysis. This helps identify initial trends and characteristics and can even refine your hypothesis. Exploratory data analysis is a crucial step in the analysis process. It is very helpful during wrangling to get in the depth of the data and hence discovering more errors to fix. Exploring the data is the main way to discover relationships among

data features, which is the preliminary step before explanatory data analysis and visualization.

4.1.6 Communicating findings

In this phase of the analysis process, we you to share what you got during the analysis as answers for your questions. The most common way to do this is via visuals and statistical results which is carried out during the explanatory analysis. In phases with exploratory visualizations, the primary audience for the visuals will be you, the analyst. The plots that have been created and demonstrated have not been particularly polished, just descriptive enough for you to gain insights into the data. On the other hand, in explanatory **analyses**, the audience will be broader. The goal will be to convey the findings to other people who do not have experience with the data. Visualizations, under this banner, should be focused on telling a specific story that an analyst wants to convey to that particular audience. Many times, these visualizations evolve from visuals created during the exploratory process, just polished up to highlight the specific intended insights. These highlights might change depending on the audience to which the findings will be presented. So, it is important that the visualizations are compelling and understandable for others.

4.2 Anaconda and Jupyter notebook

4.2.1 Anaconda

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012. As an Anaconda, Inc. product, it is also known as Anaconda Distribution or Anaconda Individual Edition, while other products from the company are Anaconda Team Edition and Anaconda Enterprise Edition, both of which are not free.

Package versions in Anaconda are managed by the package management system conda. This package manager was spun out as a separate open-source package as it ended up being useful on its own and for other things than Python. There is also a small, bootstrap

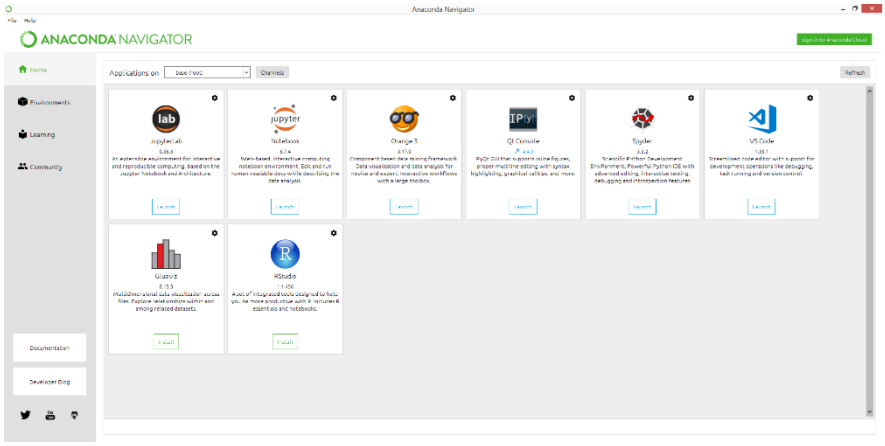
version of Anaconda called Miniconda, which includes only conda, Python, the packages they depend on, and a small number of other packages.

4.2.2 Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glue
- Orange
- RStudio
- Visual Studio Code



4.2.3 Conda:

Conda is an open source, cross-platform, language-agnostic package manager and environment management system that installs, runs, and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language (e.g., R), including multi-language projects. The conda package and environment manager is included in all versions of Anaconda, Miniconda, and Anaconda Repository.

4.2.4 Jupyter notebook

Jupyter is a free, open-source, interactive web tool known as a computational notebook, which researchers can use to combine software code, computational output, explanatory text, and multimedia resources in a single document. Computational notebooks have been around for decades, but Jupyter in particular has exploded in popularity over the past couple of years. This rapid uptake has been aided by an enthusiastic community of user–developers and a redesigned architecture that allows the notebook to speak dozens of programming languages — a fact reflected in its name, which was inspired, according to co-founder Fernando Pérez, by the programming languages Julia (Ju), Python (Py) and R.

For data scientists, that format can drive exploration. Notebooks are a form of interactive computing, an environment in which users execute code, see what happens, modify, and repeat in a kind of iterative conversation between researcher and data. They are not the only forum for such conversations — IPython, the interactive Python interpreter on which Jupyter's predecessor, IPython Notebook, was built, is another. But notebooks allow users to document those conversations, building more powerful connections between topics, theories, data, and results.

The Jupyter notebook has two components. Users input programming code or text in rectangular cells in a front-end web page. The browser then passes that code to a back-end 'kernel', which runs the code and returns the results. By Pérez's count, more than 100 Jupyter kernels have been created, supporting dozens of programming languages. Normally, each notebook can run only one kernel and one language, but workarounds exist. One demo notebook, for instance, speaks Python, Julia, R and Fortran. Researchers can also use notebooks to create tutorials or interactive manuals for their software.

4.3 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

Python 3.7 version was used for the data analysis part of this project. Four main packages were included in the code, NumPy, Pandas, Matplotlib, and Seaborn. Following we give a brief about each of them.

4.3.1 NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

NumPy

NumPy fully supports an object-oriented approach, starting, once again, with *ndarray*. For example, *ndarray* is a class, possessing numerous methods and attributes. Many of its methods are mirrored by functions in the outer-most NumPy namespace, allowing the programmer to code in whichever paradigm they prefer. This flexibility has allowed the NumPy array dialect and NumPy *ndarray* class to become the *de-facto* language of multi-dimensional data interchange used in Python.

4.3.2 Pandas

pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real-world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open-source data analysis/manipulation tool available in any language**. It is already well on its way toward this goal.

pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously typed columns, as in an SQL table or Excel spreadsheet.
- Ordered and unordered (not necessarily fixed frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels.
- Any other form of observational / statistical data sets. The data need not be labeled at all to be placed into a pandas data structure.

The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, DataFrame provides everything that R's data.frame provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

4.3.3 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

Matplotlib was originally written by John D. Hunter. Since then, it has an active development community and is distributed under a BSD-style license.

Matplotlib 2.0.x supports Python versions 2.7 through 3.10. Python 3 support started with Matplotlib 1.2. Matplotlib 1.4 is the last version to support Python 2.6. Matplotlib has pledged not to support Python 2 past 2020 by signing the Python 3 Statement.

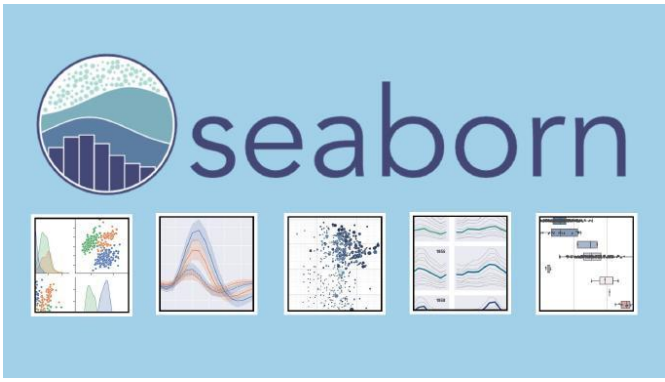
Comparison with MATLAB:

Pyplot is a Matplotlib module which provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python, and the advantage of being free and open-source.

4.3.4 Seaborn

Seaborn is a Python data visualization library based on matplotlib and integrates closely with pandas data structures. It provides a high-level interface for drawing attractive and informative statistical graphics.

Seaborn helps you explore and understand your data. Its plotting functions operate on data frames and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them. Seaborn makes it easy to switch between different visual representations by using a consistent dataset-oriented API.



Matplotlib has a comprehensive and powerful API; just about any attribute of the figure can be changed to your liking. A combination of seaborn's high-level interface and matplotlib's deep customizability will allow you both to quickly explore your data and to create graphics that can be tailored into a publication quality final product.

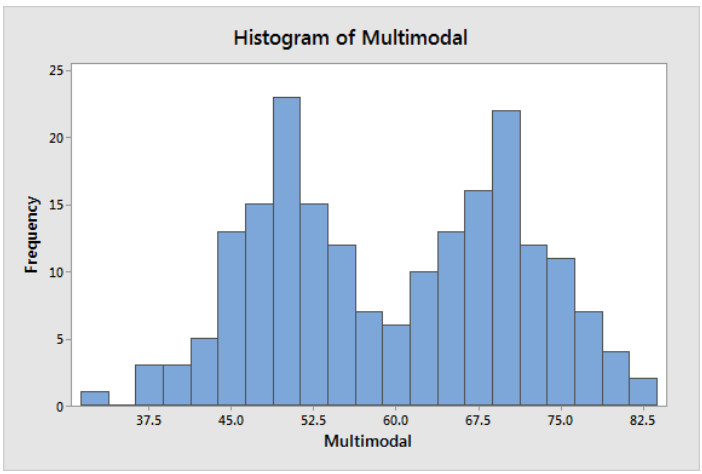
4.4 Data visuals

Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data.

This section explains the types of visuals that was used during the analysis process.

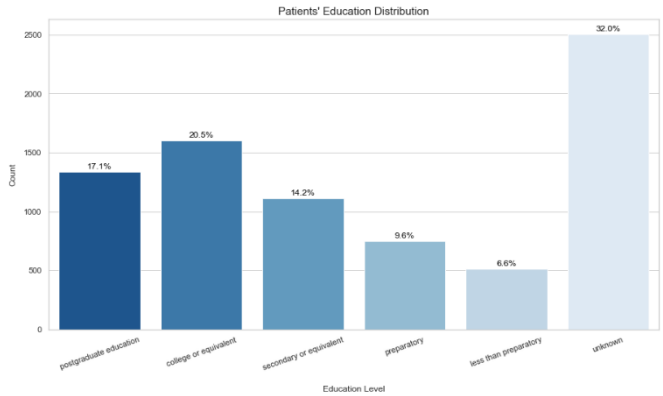
4.4.1 Histogram

A histogram is a bar graph-like representation of data that buckets a range of outcomes into columns along the x-axis. The y-axis represents the number count or percentage of occurrences in the data for each column and can be used to visualize data distributions.



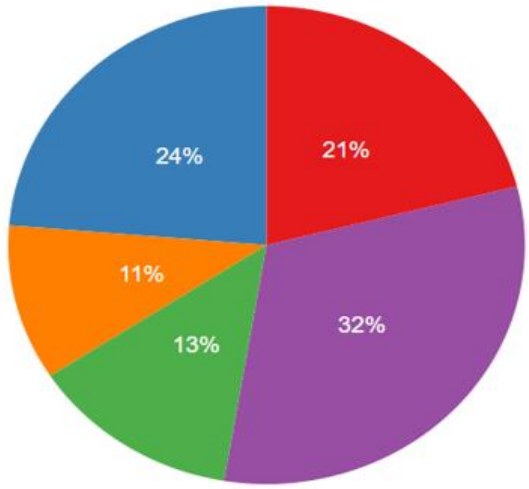
4.4.2 Bar chart

A bar chart, also known as bar graph, column chart, plots numeric values for levels of a categorical feature as bars. Levels are plotted on one chart axis, and values are plotted on the other axis. Each categorical value claims one bar, and the length of each bar corresponds to the bar's value.



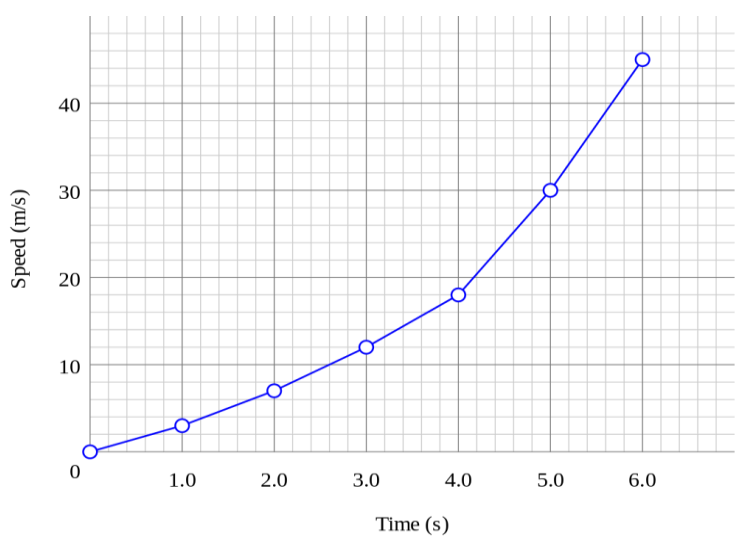
4.4.3 Pie chart

A pie chart or a circle chart is a circular statistical graphic, which is divided into slices to illustrate numerical proportion. In a pie chart, the arc length of each slice and consequently its central angle and area, is proportional to the quantity it represents.



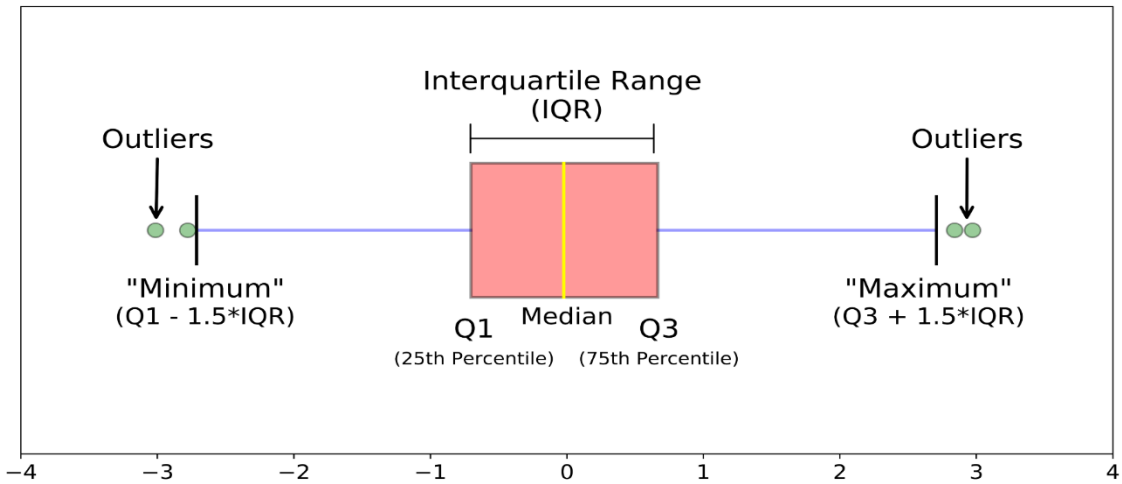
4.4.4 Line chart

A line chart or line plot or line graph or curve chart is a type of chart which displays information as a series of data points called 'markers' connected by straight line segments. It is a basic type of chart common in many fields. It is like a scatter plot except that the measurement points are ordered (typically by their x-axis value) and joined with straight line segments. A line chart is often used to visualize a trend in data over intervals of time – a time series – thus the line is often drawn chronologically. In these cases, they are known as run charts.



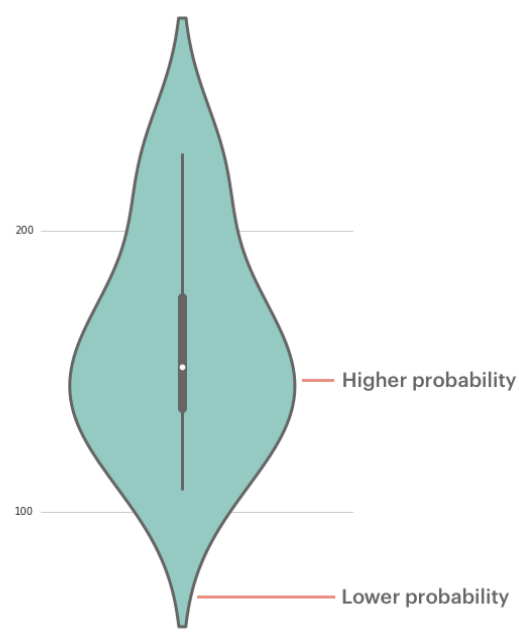
4.4.5 Box plot

A box and whisker plot—also called a box plot—displays the five-number summary of a set of data. The five-number summary is the minimum, first quartile, median, third quartile, and maximum. In a box plot, we draw a box from the first quartile to the third quartile. A vertical line goes through the box at the median.



4.4.6 Violin plot

A violin plot is a method of plotting numeric data. It is like a box plot, with the addition of a rotated kernel density plot on each side. Violin plots are like box plots, except that they also show the probability density of the data at different values, usually smoothed by a kernel density estimator.

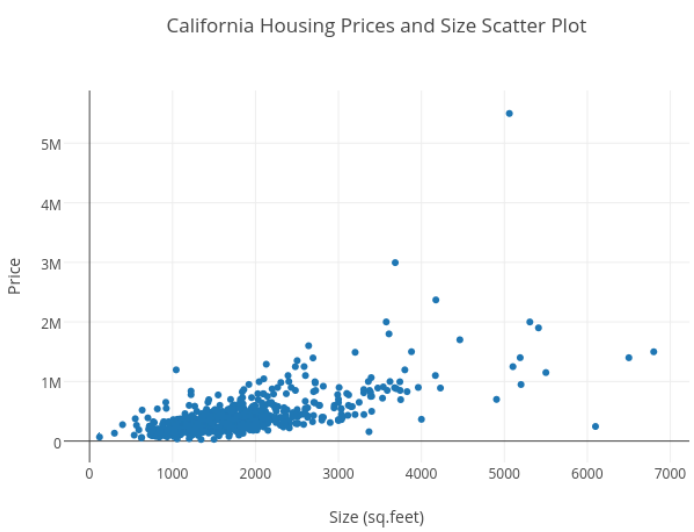


4.4.7 Heatmap

A heatmap is a graphical representation of data in two-dimension, using colors to demonstrate different factors. Heatmaps are a helpful visual aid for a viewer, enabling the quick dissemination of statistical or data-driven information.

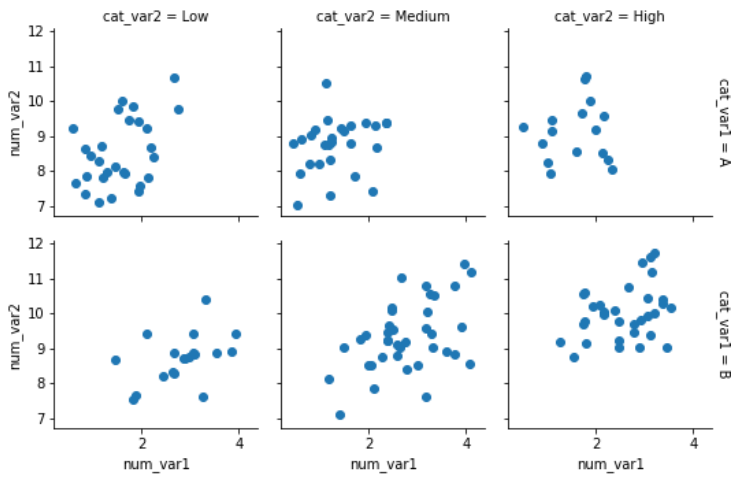
4.4.8 Scatter plot

A scatter plot (aka scatter chart, scatter graph) uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables.



4.4.9 Faceted visuals

Facets consists of multiple visualizations that allow users to see a holistic picture of their data at different granularities. Get a sense of the shape of each feature of the data using Facets Overview or explore a set of individual observations using Facets Dive.



4.5 CloCare patients' data analysis

CloCare project is composed of two major sub-projects, Electronic Health Record (EHR), and Internet of things (IOT). The EHR part of the project aims at storing patient's data so that each user has the ability to store, and thus track their medical history.

The significance of EHR is that the users can tell their medical history precisely to physicians in case they have illness or any medical emergency. This is very important for the doctors since it will help them make a proper diagnosis of the diseases.

Over time and as patient's data accumulates, we will reach a state in which the data is a good representative of the patients in a district, a city, a province, or even a country.

The analysis of this data can be very useful on a variety of levels such as, medical reports, government decisions, investment in the medical sector, etc...

In the following we will consider the type of data collected from the patients and the complete analysis process on it until reaching the findings a medical analyst seeks.

4.5.1 Patients' data

The data collected from the patients can be categorized into three main categories:

- **Personal data:** includes non-medical data (id number, name, gender, age, weight, height, education, marital status, having health insurance, the general heath condition, either a smoker or not, the number of days they work per week, Body-Mass-index, waist size, and how many drinks they have per week).
- **Diseases:** in this category, there is a number of common diseases that people usually suffer (Asthma, Congestive Heart Failure CHF, Coronary Artery Disease CAD, Heart Attack MI, Stroke CVA, Emphysema COPD, Cancer, Hypertension, Diabetes, Pulse, Systolic Blood Pressure, Diastolic Blood Pressure).
- **Medical Tests:** tests here are further categorized.
    - Complete Blood Count CBC: white blood cells count WBC, Hemoglobin HGB, Hematocrit HCT, Platelets.
    - Liver Function Test LFT: Alanine aminotransferase ALT, Aspartate aminotransferase AST, Alkaline phosphatase alk_phos.
    - Kidney Function Test KFT: Blood Urea Nitrogen BUN, Serum Creatine Cr.
    - Comprehensive Metabolic Panel CMP: Sodium, Potassium, Chloride, Bicarbonate, Calcium, Phosphorus, Total Bilirubin, Albumin, Total Protein, Globulins, Serum Glucose, Fasting Glucose,
    - Albumin-Creatine ratio (ACR)
    - Lipid Profile Test: Triglycerides, Total Cholesterol, HDL, LDL.
    - Hemoglobin A1c
    - Insulin Blood Test
    - Serum Iron Test
    - Uric Acid Test
    - Serum Cotinine Test
    - CPK isoenzymes Test
    - Lactate Dehydrogenase Test LDH
    - Spirometry: FVC, FEV1, FEV1_FVC_ratio
    - Cognitive Testing: memory

4.5.2 Dataset

Data gathering is the first step of the analysis process. The data, mentioned above, should be collected from users through a mobile application. The app will store the user's record into the database deployed on a cloud server. Then, this data will be taken to be passed through the entire analysis process.

The data collected for the project is for real patients, gathered from (data.world) website. It was a part of the American project (National Health and Nutritional Evaluation Survey 2011-2012). The data was decoded and modified to suit our project.

4.5.3 Wrangling process

The wrangling applied to the data was very conservative so that it results in no bias for the data and the findings. During this step we focused on:

- Dropping unwanted features:

```
In [3]: df.drop(['name'], axis=True, inplace=True)
```

- Removing duplicated data: dataframe.drop_duplicates()
- Changing the missing data (NAN) to (unknown) to be a separate category during the analysis:

```
In [4]: #gender
        df['gender'][df.gender.isna()] = 'unknown'

        #education
        df['education'][df.education.isna()] = 'unknown'
```

- Decoding some features so they can be interpreted by analysts and be representative to the audience.

```
# income
df['income'][df.income.isna()] = 'unknown'
df['income'][df.income == 1] = '$0 to $4,999'
df['income'][df.income == 2] = '$5k to $9,999'
df['income'][df.income == 3] = '$10k to $14,999'
df['income'][df.income == 4] = '$15k to $19,999'
df['income'][df.income == 5] = '$20k to $24,999'
df['income'][df.income == 6] = '$25k to $34,999'
df['income'][df.income == 7] = '$35k to $44,999'
df['income'][df.income == 8] = '$45k to $54,999'
df['income'][df.income == 9] = '$55k to $64,999'
df['income'][df.income == 10] = '$65k to $74,999'
df['income'][df.income == 14] = '$75k to $99,999'
df['income'][df.income == 15] = '$100k and Over'
```

- Changing categorical data into ordinal categorical data type so that they appear ordered in the findings.

```
#gen_health
Genh_levels = ['excellent', 'very good', 'good', 'fair', 'poor', 'unknown']
genh_levels = pd.api.types.CategoricalDtype(ordered=True, categories=Genh_levels)
df.gen_health = df.gen_health.astype(genh_levels)
```

- Data validation: the most important in the wrangling. Ranges were assigned for every numerical feature so as to make sure that the entered data was valid,

otherwise it is nulled. This was performed for personal numerical data like age, weight, height, and for blood pressures, and also for every numerical medical test.

```
# nulling white blood count(wbc) cells with values more 100 or less than 1
df['wbc'][df.wbc > 100] = np.nan
df['wbc'][df.wbc < 1] = np.nan

# nulling hemoglobin(hgb) cells with values more 100 or less than 1
df['hgb'][df.hgb > 25] = np.nan
df['hgb'][df.hgb < 5] = np.nan
```

- Finally, some features were corrected based on the readings of numerical data. i.e. Hypertension disease got corrected according to blood pressure values. If blood pressure values are conflicting, then we approve the hypertension case and set the pressure values to NAN.

```
# hypertension and pressure values
# First, we depend on the correct pressure values
df['hypertension'][(df.sys_bp >= 120) &(df.dia_bp >=80)] = 'yes'
df['hypertension'][(df.sys_bp < 120) & (df.sys_bp >= 90) &(df.dia_bp < 80) &(df.dia_bp >=60)] = 'no'
df['hypertension'][(df.sys_bp < 90) &(df.dia_bp < 60)] = 'hypotension'


right_hyper_ids = df['id'][((df.sys_bp >= 120) &(df.dia_bp >=80)) |
                           ((df.sys_bp < 90) &(df.dia_bp < 60)) |
                           ((df.sys_bp < 120) & (df.sys_bp >= 90) &(df.dia_bp < 80) &(df.dia_bp >=60))]

df['sys_bp'][~(df.id.isin(right_hyper_ids))] = np.nan
df['dia_bp'][~(df.id.isin(right_hyper_ids))] = np.nan
```

4.5.4 Analytics

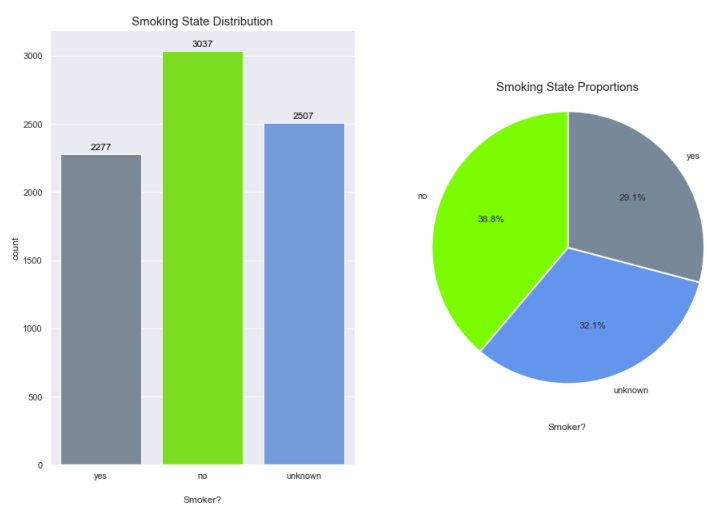
The final product of the analysis is mainly visualizations that reveals facts about the data and show the relationships among its features. The interpretations for these visualizations are the responsibility of the medical analyst who examines the required visuals to answer specific questions.

In this project, we wrote a python code to generate 260 visuals, each includes one or more plots. The visuals are categorized in four analytics for personal, diseases, and medical tests data.

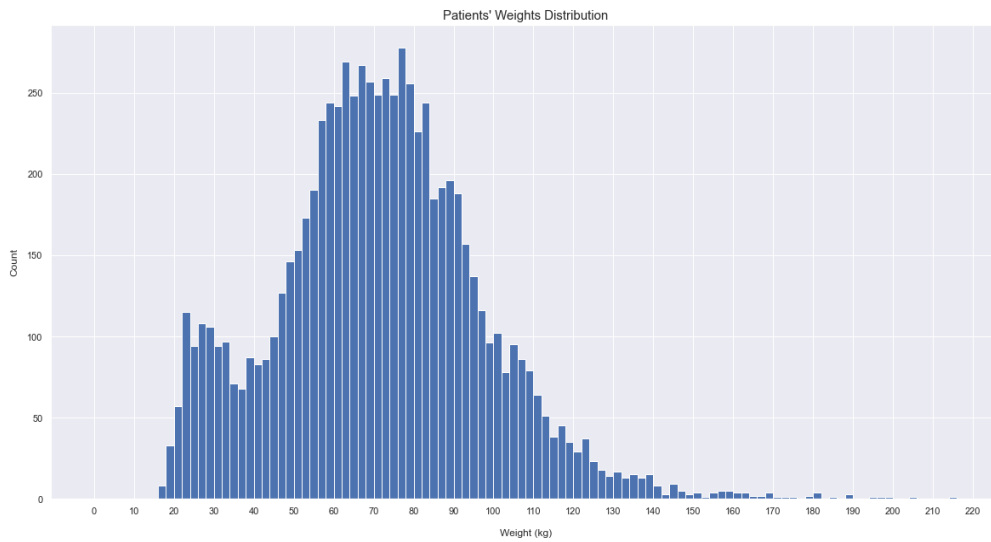
- Personal data analytics
  The personal data of the patients are analyzed on several levels. Univariate analytics for single variable distributions, bivariate analytics to show the relation between each two features, multivariate analytics for more than two features. Following is exampling these plots.
  - Single variate plots:
    This is for the smokers' proportions and counts in the data.

```
plt.figure(figsize=[15,10])
sb.set_theme(style="darkgrid")
#_____________________________first plot_____________________________
plt.subplot(1,2,1)
```

```
sb.countplot(data=df, x='smoker', palette = ['lightslategray', 'lawngreen', 'corn
flowerblue']);
smok_counts = df.smoker.value_counts()
locs, labels = plt.xticks()

# loop through each pair of locations and labels
for loc, label in zip(locs, labels):

    # get the text property for the label to get the correct count
    count = smok_counts[label.get_text()]
    pct_string = '{:.0f}'.format(count)

    # print the annotation just above the top of the bar
    plt.text(loc, count+(smok_counts[0]/100), pct_string, ha = 'center', color =
'black')

plt.title('Smoking State Distribution', fontsize= 15)
plt.xlabel('Smoker?', labelpad=20)
#_________________________________second plot_________________________________
_____
plt.subplot(1,2,2)
colors = ['lawngreen', 'cornflowerblue', 'lightslategray']

plt.pie(x=smok_counts, labels=smok_counts.index, startangle=90, counterclock=True
, colors=colors,
        autopct=lambda p: '{:.1f}%'.format(p),
        wedgeprops = { 'linewidth' : 2, 'edgecolor' : 'white' })

plt.axis('square')
plt.xlabel('Smoker?', labelpad=30)
plt.title('Smoking State Proportions', fontsize= 15);
```



```
plt.figure(figsize=[20, 10])
sb.set_theme(style="darkgrid")

bins = np.arange(0, df.weight_kg.max()+2, 2)
ticks = np.arange(0, df.weight_kg.max()+10, 10)
labels = ['{:.0f}'.format(v) for v in ticks]
plt.hist(data=df, x='weight_kg', bins= bins)
plt.xticks(ticks, labels)

plt.title("Patients' Weights Distribution", fontsize= 15)
plt.ylabel('Count', labelpad=10)
plt.xlabel('Weight (kg)', labelpad=15);
```

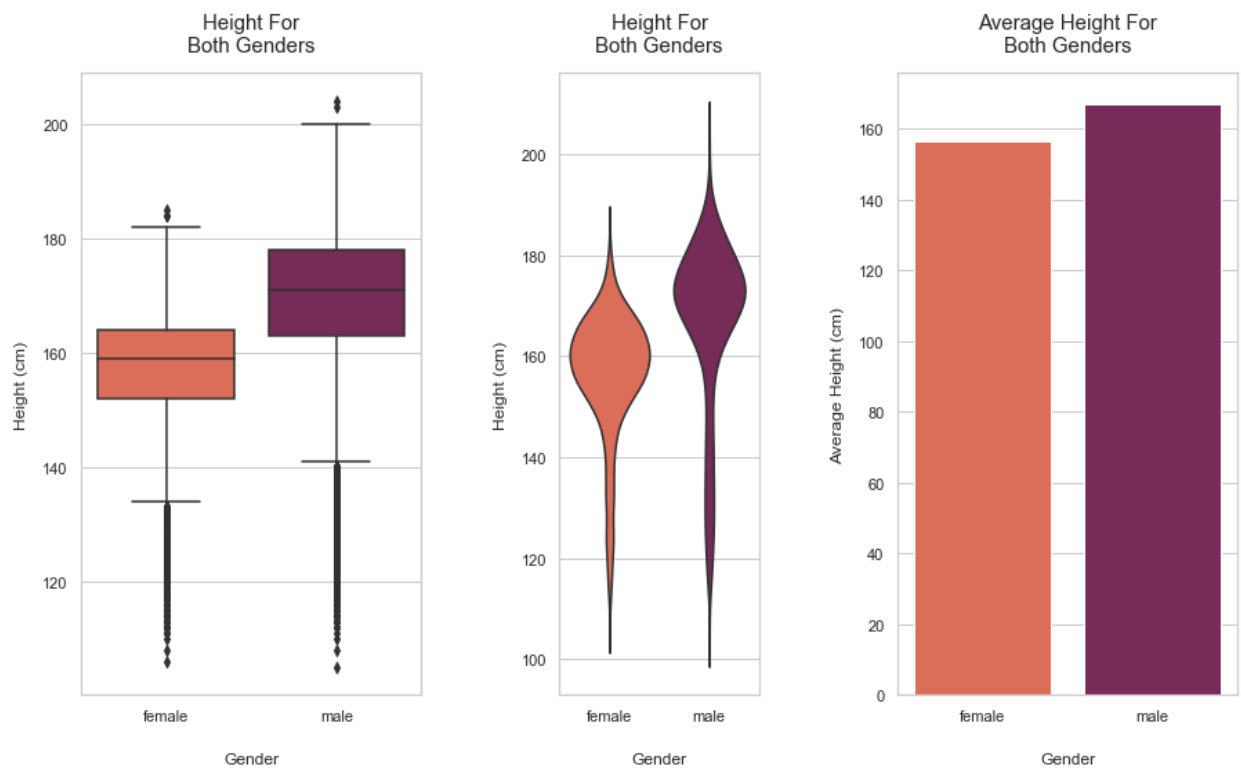


- Bivariate visuals

  This visual shows the age distribution and average age for both genders.

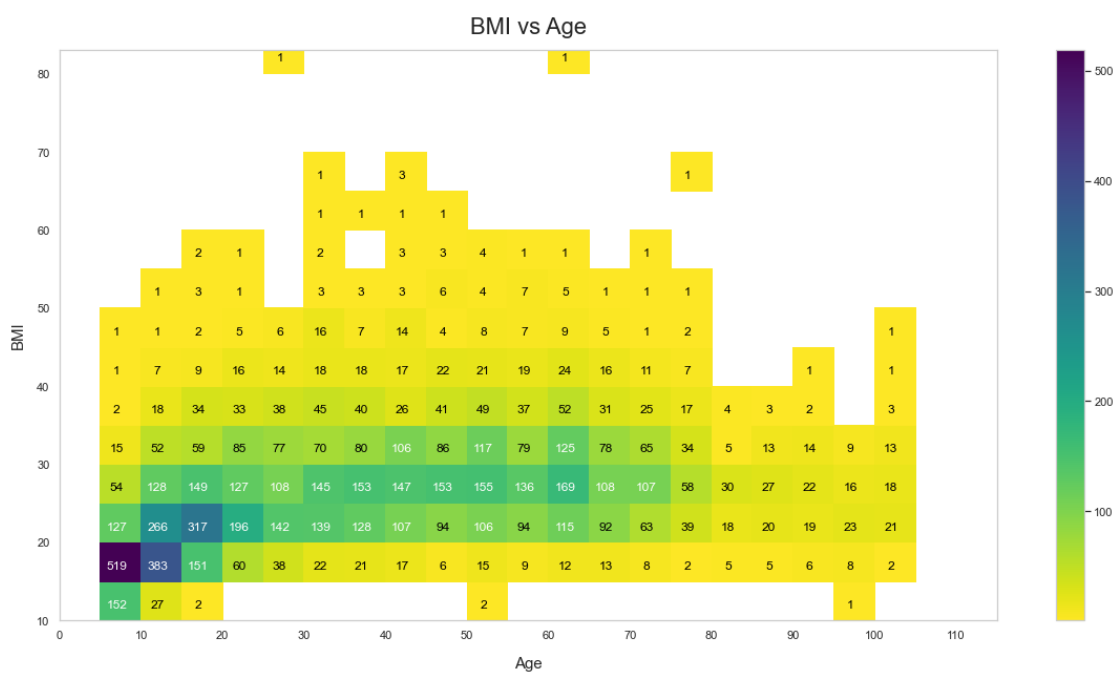
```
plt.figure(figsize=[15, 8])
sb.set_theme(style="whitegrid")

#_____________________________first plot_____________________________
plt.subplot(1,3,1)
sb.boxplot(data=df, x='gender', y='height_cm', order=['female', 'male'], palette=
'rocket_r')
plt.title('Height For\nBoth Genders', fontsize= 15, pad=15)
plt.ylabel('Height (cm)', fontsize=12, labelpad=10);
plt.xlabel('Gender', fontsize=12, labelpad=20)

#_____________________________second plot_____________________________
plt.subplot(1,5,3)
sb.violinplot(data=df, x='gender', y='height_cm', inner=None, order=['female', 'm
ale'], palette='rocket_r')
```

```
plt.title('Height For\nBoth Genders', fontsize= 15, pad=15)
plt.ylabel('Height (cm)', fontsize=12, labelpad=10);
plt.xlabel('Gender', fontsize=12, labelpad=20)

#_____________________________second plot_____________________________
plt.subplot(1,3,3)
sb.barplot(data=df, x='gender', y='height_cm', errwidth=0, order=['female', 'male
'], palette='rocket_r')

plt.title('Average Height For\nBoth Genders', fontsize= 15, pad=15)
plt.xlabel('Gender', fontsize=12, labelpad=20)
plt.ylabel('Average Height (cm)', fontsize=12, labelpad=10);
```



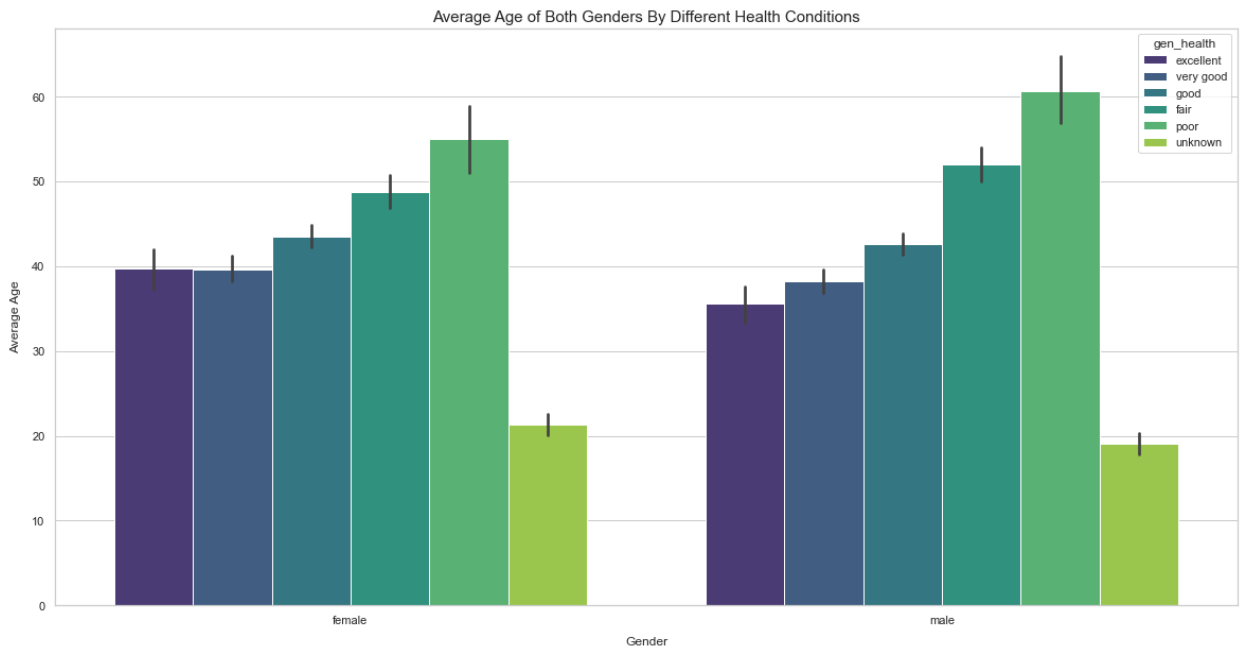

```
plt.figure(figsize=[20, 10])
sb.set_theme(style="whitegrid")

x_bins = np.arange(0, 120, 5)
y_bins = np.arange(10, 110, 5)

heat_map = plt.hist2d(data=df, x='age', y='bmi', cmin=0.5, cmap='viridis_r', bins
=[x_bins, y_bins])
plt.colorbar()
```

```
counts = heat_map[0]

for i in range(counts.shape[0]):
    for j in range(counts.shape[1]):

        c = counts[i,j]
        if c>= 100 :
            plt.text(x_bins[i]+2, y_bins[j]+2, int(c), ha='center', va='center',
color='white');

        elif c>0 :
            plt.text(x_bins[i]+2, y_bins[j]+2, int(c), ha='center', va='center',
color='black');


x_bins = np.arange(0, 120, 10)
labels_x = ['{:.0f}'.format(v) for v in x_bins]
y_bins = np.arange(10, df.bmi.max()+5, 10)
labels_y = ['{:.0f}'.format(v) for v in y_bins]
plt.xticks(x_bins, labels_x)
plt.yticks(y_bins, labels_y)
plt.ylim(10, df.bmi.max()+1)

plt.title('BMI vs Age', fontsize= 22, pad=15)
plt.xlabel('Age', fontsize= 15, labelpad=15)
plt.ylabel('BMI', fontsize= 15, labelpad=10)
```



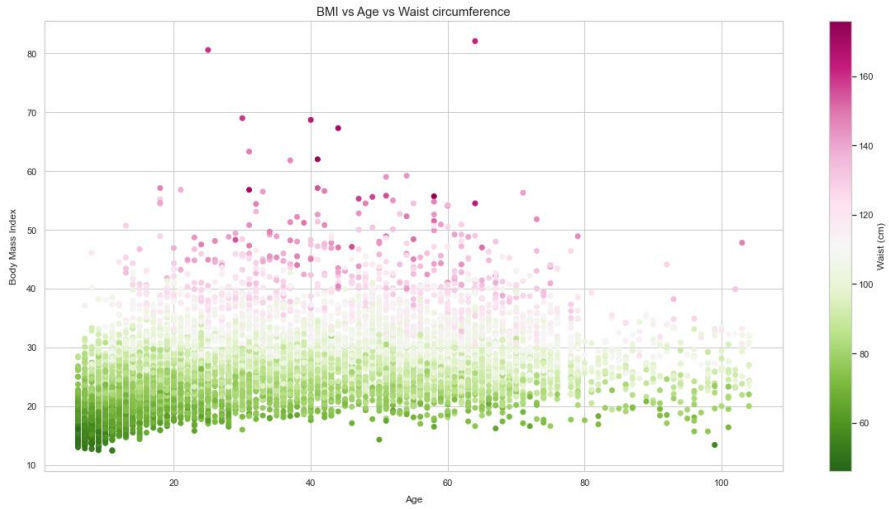
- Multivariate visuals
  This visual shows average age vs gender for each health condition.

```
plt.figure(figsize=[20,10])

sb.barplot(data=df, x='gender', order=['female', 'male'], y='age', hue='gen_healt
h', palette='viridis', alpha=1)

plt.title('Average Age of Both Genders By Different Health Conditions', fontsize=
 15)
plt.ylabel('Average Age', labelpad=10)
plt.xlabel('Gender', labelpad=10);
```



This visual shows the age vs body-mass-index vs waist circumference size.

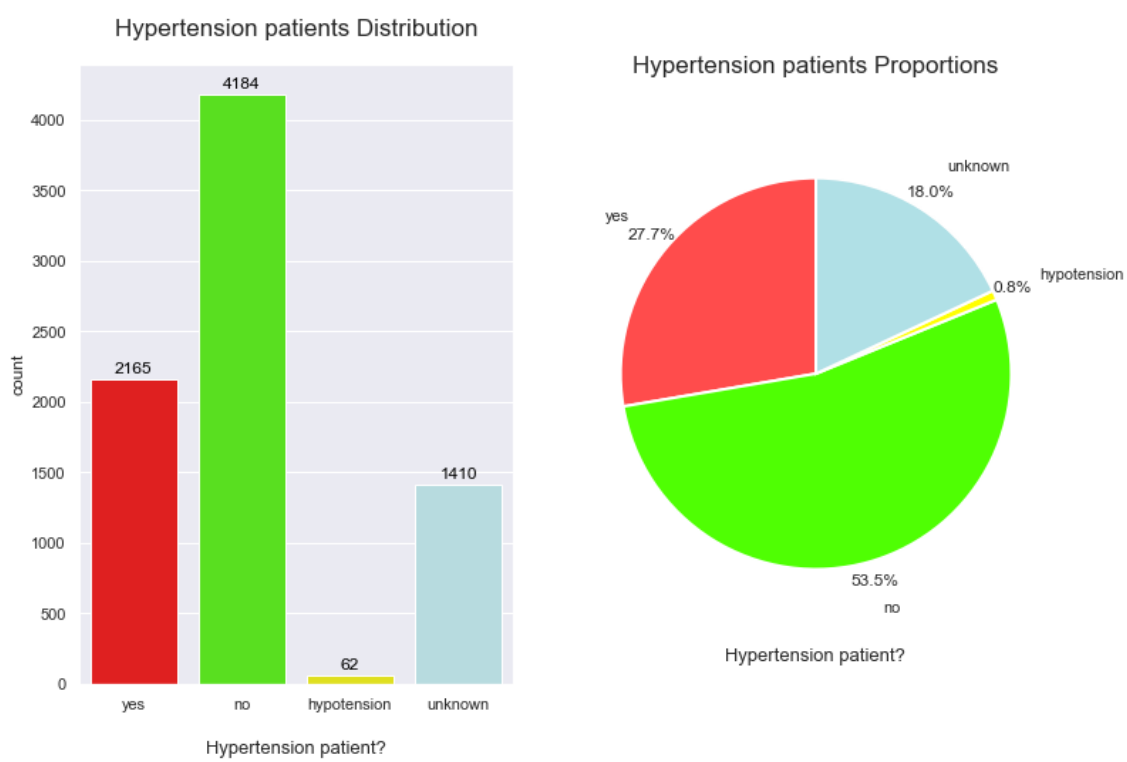
```
plt.figure(figsize=[20, 10])
plt.scatter(data=df, x='age', y='bmi', c='waist_cm', cmap='PiYG_r', alpha=1)
plt.colorbar(label='Waist (cm)');

plt.title('BMI vs Age vs Waist circumference', fontsize= 15)
plt.ylabel('Body Mass Index', labelpad=10)
plt.xlabel('Age', labelpad=10);
```



- Diseases analytics
  Diseases analytics mainly show the percentages and counts of patients associated with each disease or multiple correlated diseases. The following visual shows the hypertension disease among the patients.

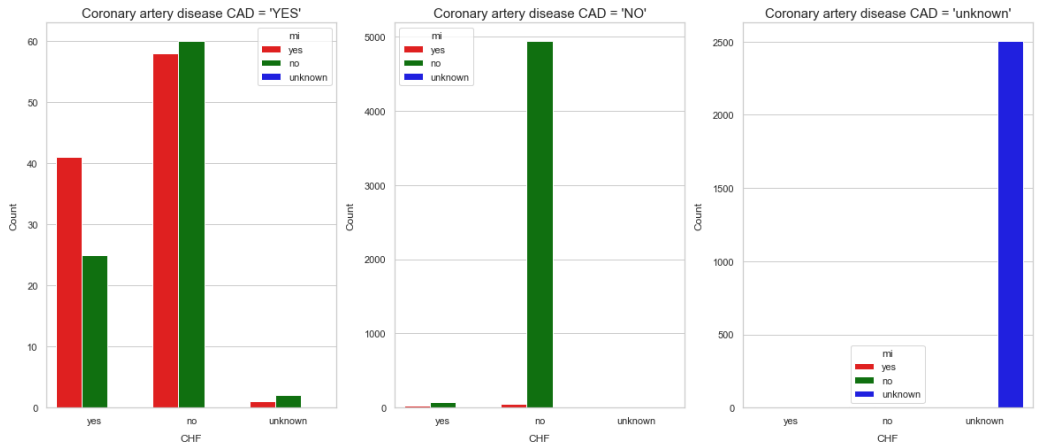
```
plt.figure(figsize=[12,8])
sb.set_theme(style="darkgrid")

#_____________________________first plot_____________________________________
plt.subplot(1,2,1)

sb.countplot(data=df, x='hypertension', palette = [(1,0,0,.7), (.3,1,0,.99), 'yel
low', 'powderblue']);
_counts = df.hypertension.value_counts()
locs, labels = plt.xticks()

# loop through each pair of locations and labels
for loc, label in zip(locs, labels):

    # get the text property for the label to get the correct count
    count = _counts[label.get_text()]
    count_string = '{}'.format(count)

    # print the annotation just above the top of the bar
    plt.text(loc, count+(_counts[0]/100), count_string, ha = 'center', color = 'b
lack') #, palette=()

plt.title('Hypertension patients Distribution', fontsize= 17, pad=20)
plt.xlabel('Hypertension patient?', fontsize= 13, labelpad=20)
```

```
#_____________________________________second plot_____________________________________
_____
plt.subplot(1,2,2)

colors = [(1,0,0,.7), (.3,1,0,.99), 'yellow', 'powderblue']
_counts = _counts.sort_index()

if (round(_counts.values.min()*100 / df.shape[0], 1) < 5):
    labels_dis = 1.25
    prc_dis = 1.1
else:
    labels_dis = 1.1
    prc_dis = .7

plt.pie(x=_counts, labels=_counts.index, startangle=90, counterclock=True, colors
=colors,
        autopct=lambda p: '{:.1f}%'.format(p),
        wedgeprops = { 'linewidth' : 2, 'edgecolor' : 'white' }, pctdistance=prc_
dis, labeldistance=labels_dis)#, textprops={'color':'white'}

plt.axis('square')

plt.title('Hypertension patients Proportions', fontsize= 17, pad=60)
plt.xlabel('Hypertension patient?', fontsize= 13, labelpad=40);
```



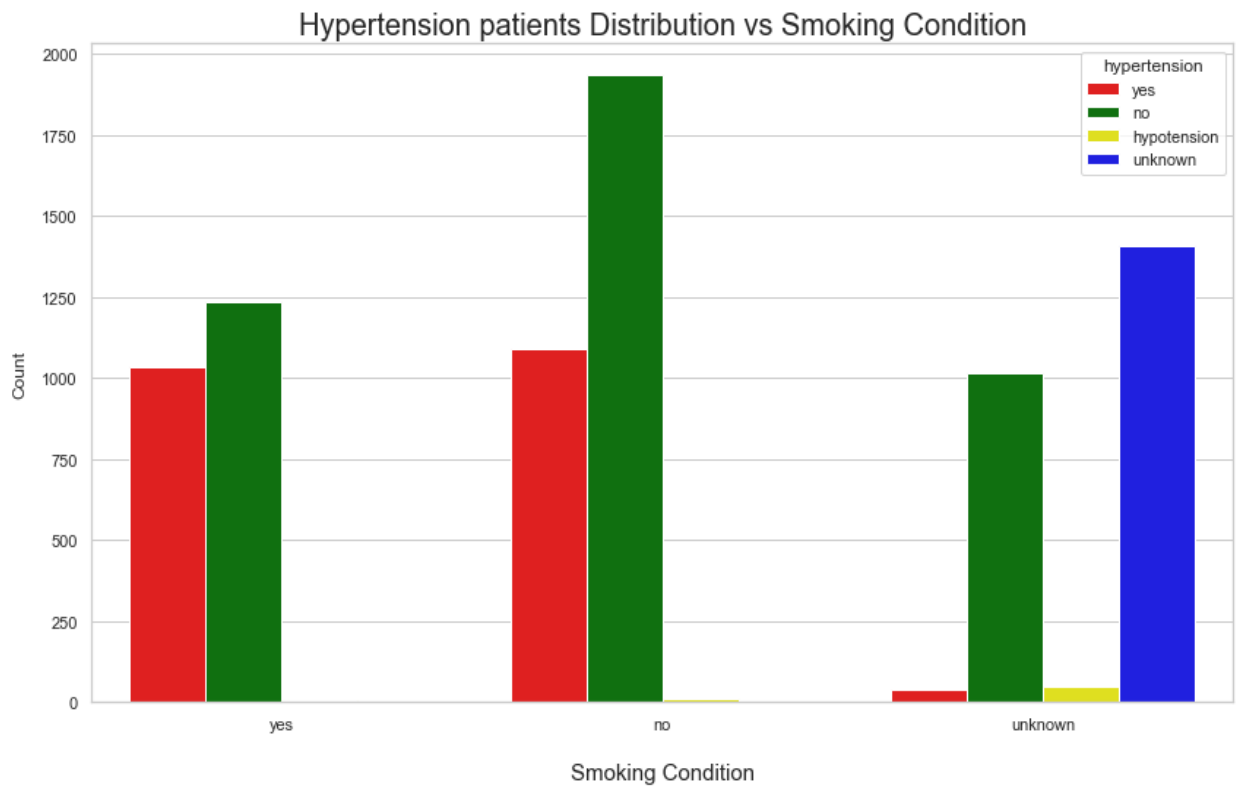
The following plot relates the common heart diseases (CHF, CAD, MI).

```
plt.figure(figsize=[20, 8])
sb.set_theme(style='whitegrid')

plt.subplot(1,3,1)
sb.countplot(data=df[(df.cad == 'yes')], x='chf', order=['yes', 'no', 'unknown'],
 hue='mi', palette=['red', 'green', 'blue'])

plt.xticks(rotation=0)
plt.title("Coronary artery disease CAD = 'YES'", fontsize= 15)
plt.ylabel('Count', labelpad=10)
plt.xlabel('CHF', labelpad=10);

# ------------------------------------------------
plt.subplot(1,3,2)
sb.countplot(data=df[(df.cad == 'no')], x='chf', order=['yes', 'no', 'unknown'],
hue='mi', palette=['red', 'green', 'blue'])

plt.xticks(rotation=0)
plt.title("Coronary artery disease CAD = 'NO'", fontsize= 15)
plt.ylabel('Count', labelpad=10)
plt.xlabel('CHF', labelpad=10);

# ------------------------------------------------
plt.subplot(1,3,3)
sb.countplot(data=df[(df.cad == 'unknown')], x='chf', order=['yes', 'no', 'unknow
n'], hue='mi', palette=['red', 'green', 'blue'])

plt.xticks(rotation=0)
plt.title("Coronary artery disease CAD = 'unknown'", fontsize= 15)
plt.ylabel('Count', labelpad=10)
plt.xlabel('CHF', labelpad=10);
```



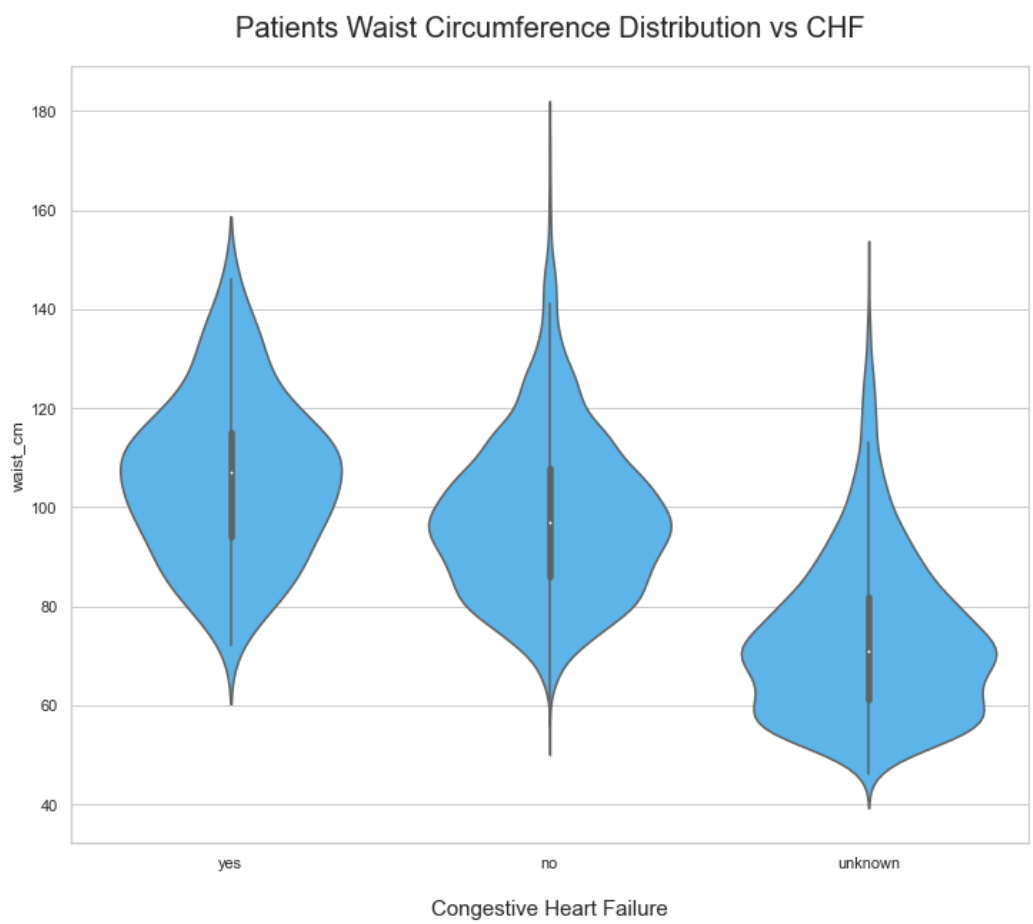
- Diseases and personal data analytics
  Visuals here shows how the personal features affect the existence of a disease. The following figure shows the hypertension vs smoking condition.

```
plt.figure(figsize=[14, 8])
sb.set_theme(style='whitegrid')

sb.countplot(data=df, x='smoker', hue='hypertension', palette=['red', 'green', 'y
ellow', 'blue'])

plt.xticks(rotation=0)
plt.title("Hypertension patients Distribution vs Smoking Condition", fontsize= 20
)
plt.ylabel('Count', labelpad=10)
plt.xlabel('Smoking Condition', fontsize=15, labelpad=20);
```



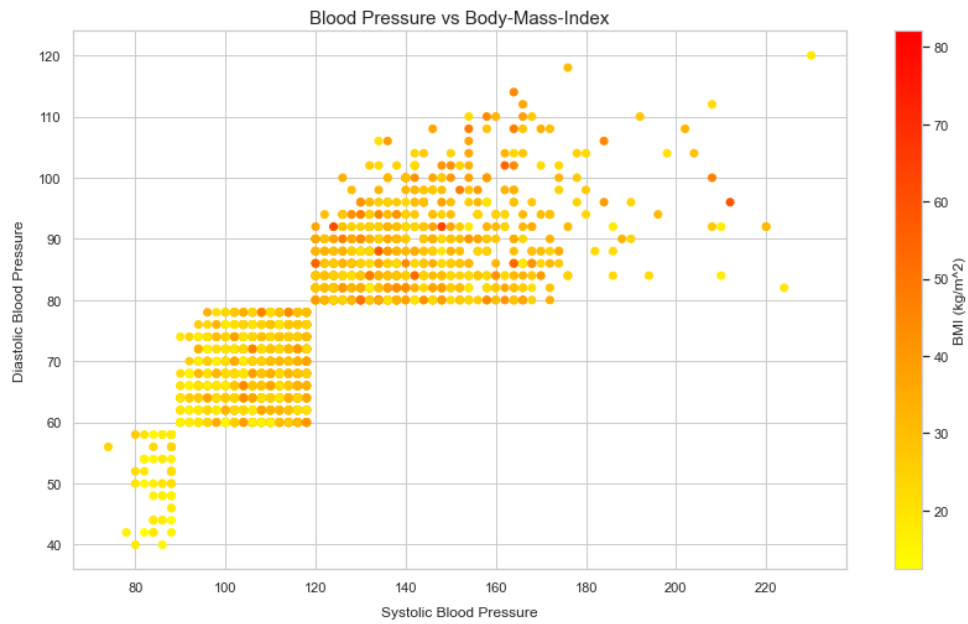
The following figure shows the relation between the waist size and cognitive heart failure disease.

```
plt.figure(figsize=[12,10])
sb.set_theme(style='whitegrid')

sb.violinplot(data=df, x='chf', y='waist_cm', color=sb.color_palette('cool', 10)[
2])

plt.title('Patients Waist Circumference Distribution vs CHF', fontsize=20, pad=20
)
plt.xlabel('Congestive Heart Failure', fontsize=15, labelpad=20);
```



The following plot shows the correlation between body-mass-index and blood pressures.

```
plt.figure(figsize=[14, 8])
plt.scatter(data=df, x='sys_bp', y='dia_bp', c='bmi', cmap='autumn_r', alpha=1)
plt.colorbar(label='BMI (kg/m^2)');

plt.title('Blood Pressure vs Body-Mass-Index', fontsize= 15)
plt.ylabel('Diastolic Blood Pressure', labelpad=10)
plt.xlabel('Systolic Blood Pressure', labelpad=10);
```

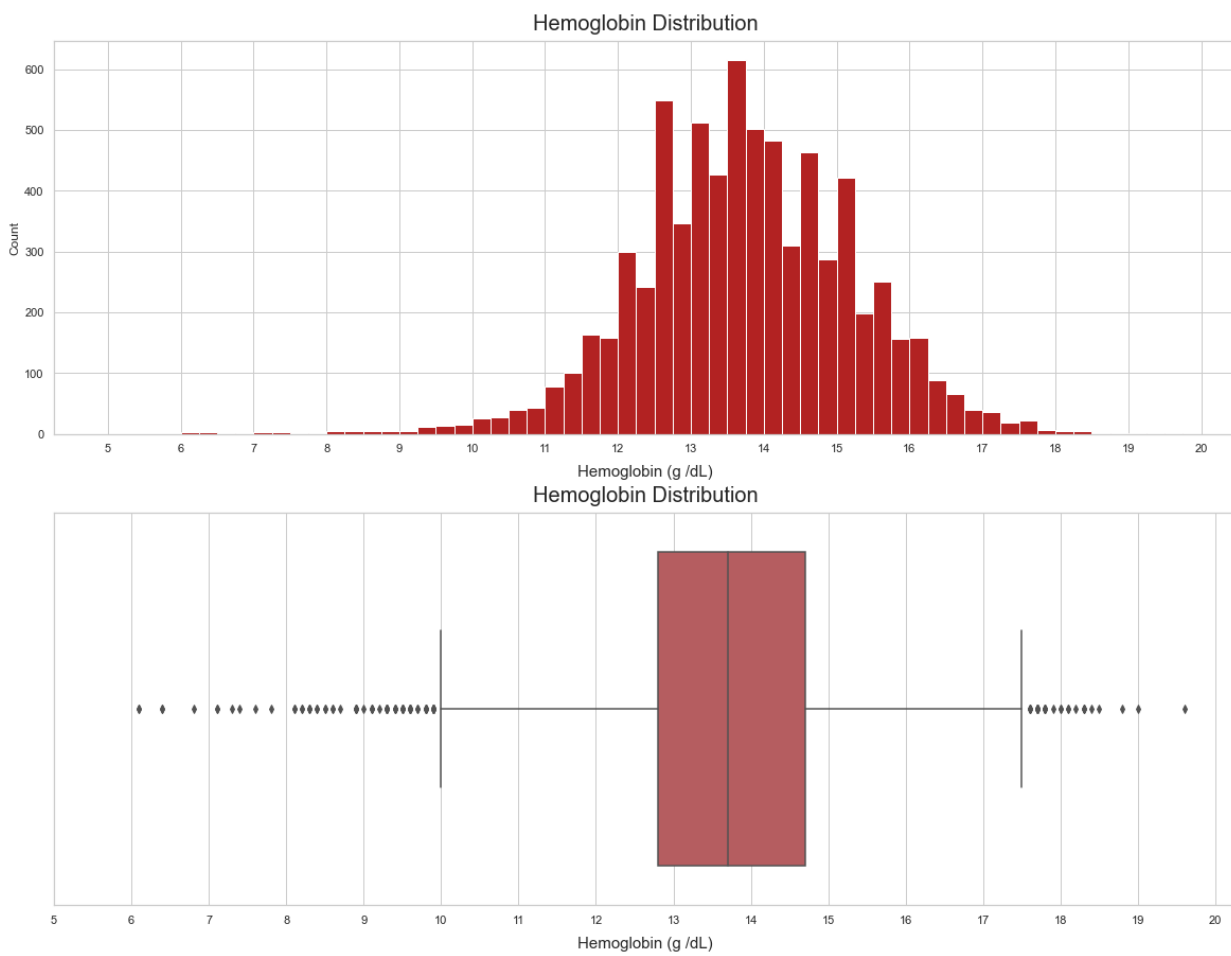


- Medical tests analytics
  Several medical tests are included in the analysis. The analysis mainly focused on the distribution of the features existing, the distributions for both genders, the relations between the correlated features, and the relations between some features and the correlated diseases.
  Here we consider Hemoglobin as an example.

```
plt.figure(figsize=[20,15])
sb.set_theme(style=None)

plt.subplot(2,1,1)
bins = np.arange(5, df.hgb.max()+.25, .25)
plt.hist(data=df, x='hgb', bins= bins, color='firebrick')

ticks = np.arange(5, df.hgb.max()+1, 1)
labels = ['{:.0f}'.format(v) for v in ticks]
```

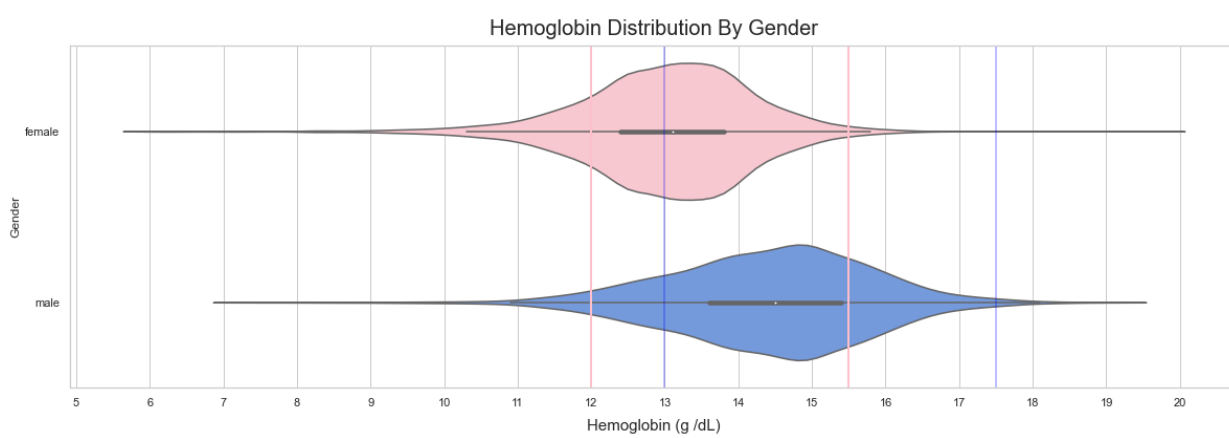
```
plt.xticks(ticks, labels)
plt.title('Hemoglobin Distribution', pad=10, fontsize=20)
plt.xlabel('Hemoglobin (g /dL)', labelpad=10, fontsize=15)
plt.ylabel('Count');


plt.subplot(2,1,2)
sb.boxplot(data=df, x='hgb', color='r')

ticks = np.arange(5, df.hgb.max()+1, 1)
labels = ['{:.0f}'.format(v) for v in ticks]
plt.xticks(ticks, labels)
plt.title('Hemoglobin Distribution', pad=10, fontsize=20)
plt.xlabel('Hemoglobin (g /dL)', labelpad=10, fontsize=15)
plt.ylabel('');
```

Hemoglobin distribution:



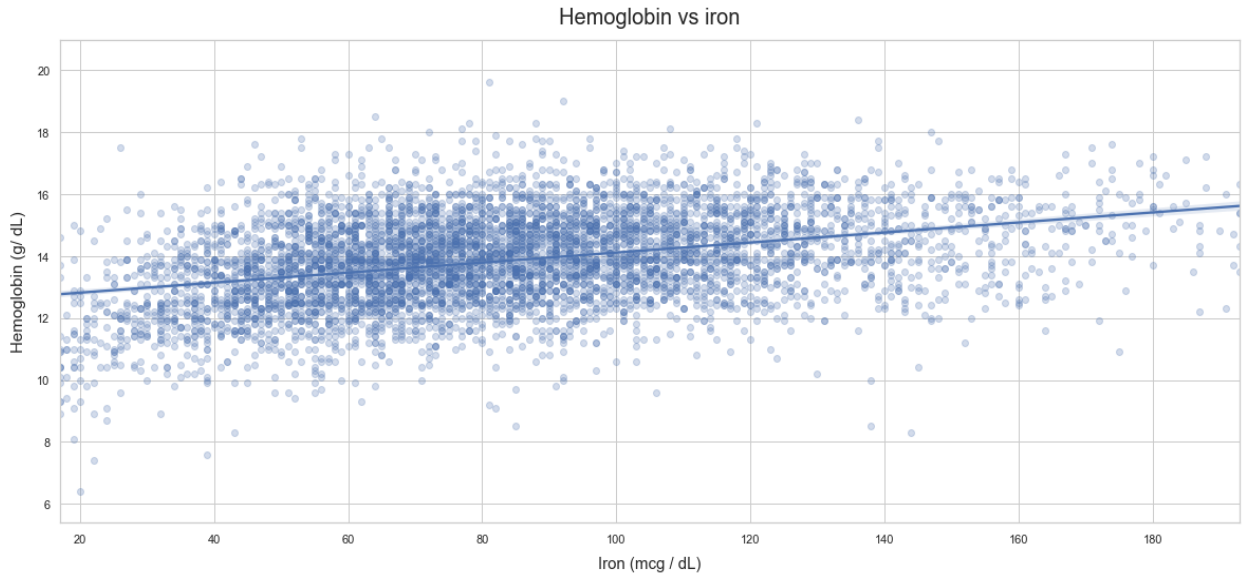
Hemoglobin distribution for both genders:

```
plt.figure(figsize=[20,6])
sb.set_theme(style=None)

sb.violinplot(data=df, x='hgb', y='gender', order=['female', 'male'], orient='hor
izontal', palette=['pink', 'cornflowerblue']);

plt.axvline(x=12, linestyle='-', linewidth=2, color='pink', alpha=1)
plt.axvline(x=15.5, linestyle='-', linewidth=2, color='pink', alpha=1)
plt.axvline(x=13.0, linestyle='-', linewidth=2, color='blue', alpha=.25)
plt.axvline(x=17.5, linestyle='-', linewidth=2, color='blue', alpha=.25)


ticks = np.arange(5, df.hgb.max()+1, 1)
labels = ['{:.0f}'.format(v) for v in ticks]
plt.xticks(ticks, labels)
plt.title('Hemoglobin Distribution By Gender', pad=10, fontsize=20)
plt.xlabel('Hemoglobin (g /dL)', labelpad=10, fontsize=15)
plt.ylabel('Gender');
```



The correlation between Hemoglobin and iron in serum:

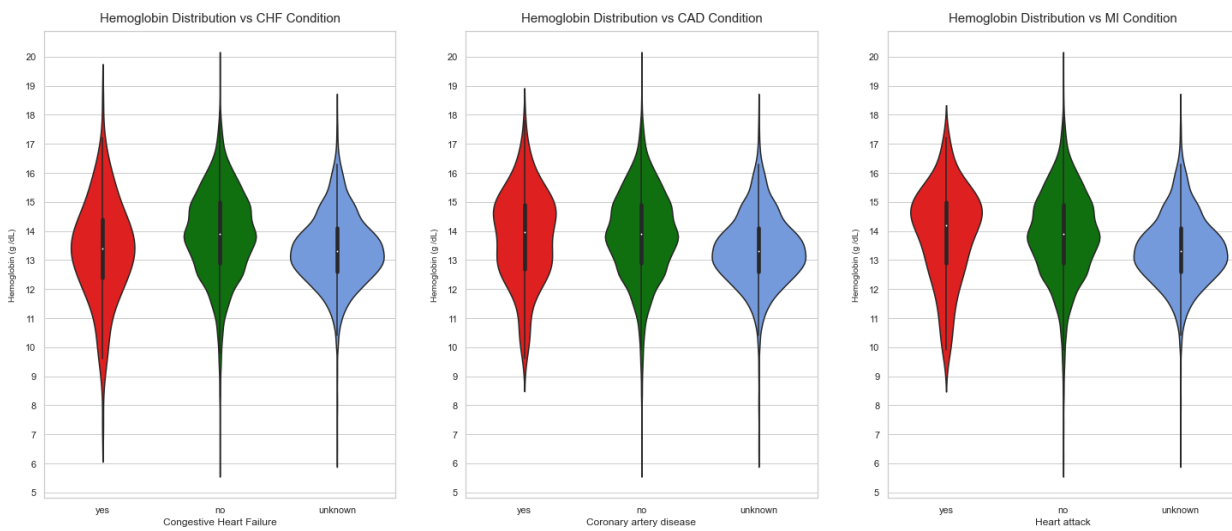
```
plt.figure(figsize=[20,8])
sb.set_theme(style=None)
sb.regplot(data=df, x='iron', y='hgb', scatter_kws={'alpha':.25}, truncate=False)
plt.title('Hemoglobin vs iron', fontsize=20, pad=15)
plt.xlabel('Iron (mcg / dL)', fontsize=15, labelpad=10)
plt.ylabel('Hemoglobin (g/ dL)', fontsize=15, labelpad=10);

plt.xlim(df.iron.quantile(.01), df.iron.quantile(.99));
```



The relation between Hemoglobin heart diseases:

```
plt.figure(figsize=[25,10])
sb.set_theme(style=None)

plt.subplot(1,3,1)
sb.violinplot(data=df, y='hgb', x='chf', palette=['red', 'green', 'cornflowerblue
']);

ticks = np.arange(5, df.hgb.max()+1, 1)
labels = ['{:.0f}'.format(v) for v in ticks]
plt.yticks(ticks, labels)
plt.title('Hemoglobin Distribution vs CHF Condition', pad=10, fontsize=15)
plt.ylabel('Hemoglobin (g /dL)', labelpad=10, fontsize=10)
plt.xlabel('Congestive Heart Failure');

plt.subplot(1,3,2)
sb.violinplot(data=df, y='hgb', x='cad', palette=['red', 'green', 'cornflowerblue
']);

ticks = np.arange(5, df.hgb.max()+1, 1)
labels = ['{:.0f}'.format(v) for v in ticks]
plt.yticks(ticks, labels)
plt.title('Hemoglobin Distribution vs CAD Condition', pad=10, fontsize=15)
plt.ylabel('Hemoglobin (g /dL)', labelpad=10, fontsize=10)
plt.xlabel('Coronary artery disease');
```

```
plt.subplot(1,3,3)
sb.violinplot(data=df, y='hgb', x='mi', palette=['red', 'green', 'cornflowerblue'
]);

ticks = np.arange(5, df.hgb.max()+1, 1)
labels = ['{:.0f}'.format(v) for v in ticks]
plt.yticks(ticks, labels)
plt.title('Hemoglobin Distribution vs MI Condition', pad=10, fontsize=15)
plt.ylabel('Hemoglobin (g /dL)', labelpad=10, fontsize=10)
plt.xlabel('Heart attack');
```



Conclusion

We have different layers in architecture, at the bottom, we have the embedded systems level that assigns to the devices itself. The design follows layered architecture scheme, it has a bootloader with OTA update capability running on stm32 – are based microcontroller, modules for different sensors and user interface.

The devices are connected in the hospital using Wired/Wireless connections to the cloud through the internet.

At the cloud end, data analysis is performed on data gathered from different facilities to derive meaningful information and get statistics about a certain area or a life stage. As we're implementing our own cloud, it won't be running 24/7, so we also offer cloud service providing.

We use IoT technology to approach our solution like using http/s protocol and Wi-Fi.

- Development cycle is being processed in parallel.
    - In the embedded systems we start at the bottom by developing a bootloader with OTA ability, followed by designing and implementing the drivers for various medical sensors using the layered architecture scheme, then we start with the RFID module, and finally the Wi-Fi communication layer to process data between cloud and devices in the final integration phase.
    - In the cloud and server end, we start by building up the basic components of the cloud using Openstack and Ovirt, followed by a web application on the cloud. Finally, develop a cloud service provider to utilize cloud resources.
    - In the mobile application we start by making a UI/UX design using Adobe XD then implement it using Flutter to work for

both Android and IOS, we then make APIs that we can later integrate in our application to access the database we created in the backend, we will also create a website that has the same functionality as the mobile application (Doctor registration, access patients' profiles or create new one or add new records to an existing profile and more)

- Quality assurance
  Individual components of the system are periodically tested as units. Another test is carried out at the integration phase, and finally with the prototype ready we perform final testing by gathering feedback from beta-users.
- Solution deployment
  Deployment of the project includes installing devices in the medical organizations, training medical staff on the devices and the mobile application, and getting the cloud up and running 24/7.

Future Work

In bootloader design we intend to add a backup version functionality to the system, improve its ability to recover from failures, and add a better reset mechanism rather than a reset button.

The registration device application needs to be enhanced in terms of security and stability.

The monitoring device sensors will be replaced by a real monitoring device found in hospitals and data logged out of it.

The data analysis process will be improved to include AI and Machine learning techniques. These techniques will improve the wrangling process, specially dealing with missing data, so that we can expect these data using what is provided from the dataset.

The analysis can be improved to include regional, timing, and age-clustered analysis. In addition to improving the wrangling and the analysis, the visualizations will be developed to include more visuals.

Project Team Members

Project name

Digitalizing health care systems using IoT solutions. (CloCare)

Team members	Academic year
Nourhan Mansour	4th year - CSED
Eman Ashraf	4th year - CSED
Sarah Salem	4th year - CSED
Omar El-Sheikh	4th year - CSED
Muhammed Hossam	4th year - CSED
Ahmed El-Gnedy	4th year – CSED
Ahmed Mogahed	4th year – CSED
Islam Mohab	4th year – CSED

Project supervisors:

Dr/ Mahmoud Safaan

Eng/ Azzahraa Abdul-Aleim

References

1. https://docs.openstack.org/
2. https://www.terraform.io/
3. https://docs.docker.com/
4. https://kubernetes.io/docs/home/
5. https://rancher.com/docs/
6. https://helm.sh/docs/intro/
7. https://github.com/bitnami/charts
8. https://docs.ansible.com/ansible_community.html
9. https://docs.python.org/3/
10. https://www.shellscript.sh/
11. https://www.beningo.com/wp-content/uploads/images/Papers/bootloader_design_for_microcontrollers_in_embedded_systems%20.pdf
12. http://www.ocfreaks.com/create-new-stm32-project-keil-uvision-5-tutorial/
13. https://www.freertos.org/
14. https://www.st.com/resource/en/datasheet/stm32f103c8.pdf
15. https://www.topwaydisplay.com/sites/default/files/2021-01/HMT070ETD-1D.pdf
16. https://datasheets.maximintegrated.com/en/ds/MAX30102.pdf
17. https://github.com/clocare/
18. https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library/
19. https://github.com/ninharp/STM32F103_RFID_Access_Control/
20. https://docs.espressif.com/
21. https://www.anaconda.com/
22. https://data.world/
23. https://stackoverflow.com/
24. Udacity Nano Degree In Data Analysis https://www.udacity.com/
25. https://themeforest.net
26. https://www.w3schools.com
27. Udemy Front-End course.
28. https://github.com/piotr-jura-udemy

29. https://www.php.net/docs.php
30. https://laravel.com/docs/5.7
31. https://flutter.dev/docs
32. https://dart.dev/guides
33. https://material.io/design