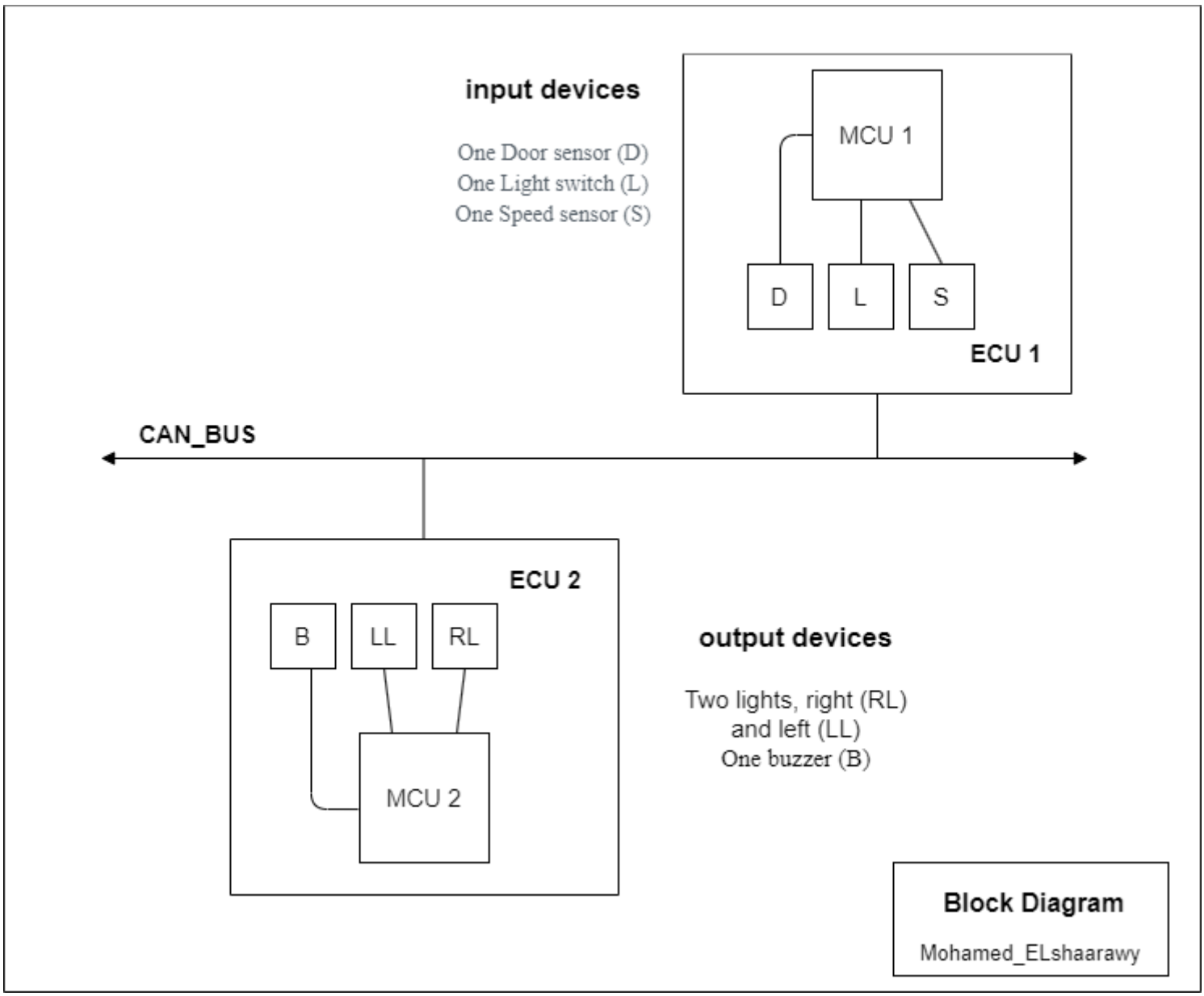
Automotive door control system design

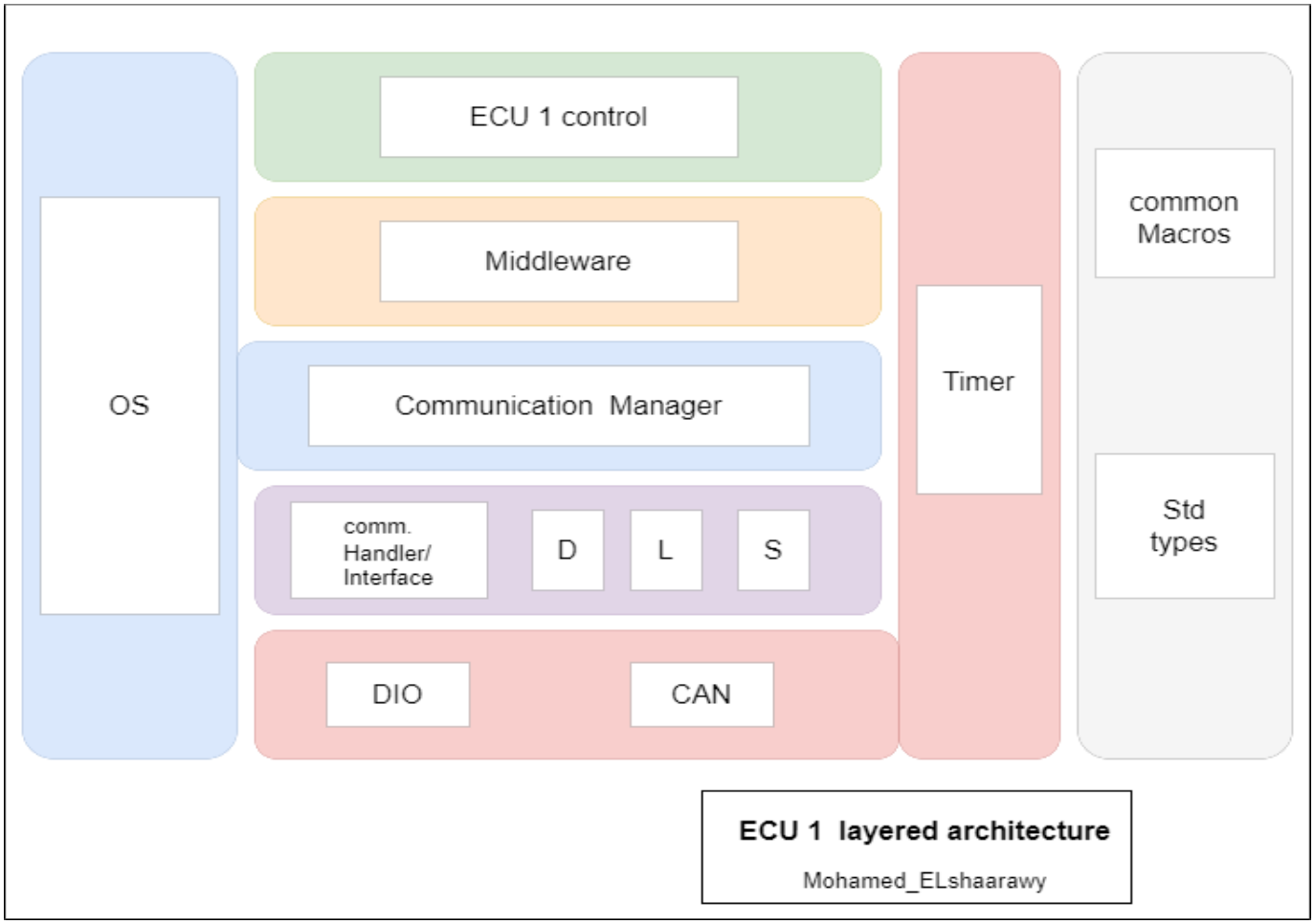
Static design analysis

- Block Diagram



For ECU 1:

1. Make the layered architecture



2. Specify ECU components and modules

- Application (ECU 1 control) : Where system logic and tasks are implemented.
  - Main.c
- Middleware : For routing the system to its desired destination.
  - Middleware.c , Middleware.h
- Services:
  - OS : assume (FreeRTOS)
    . all Kernel files
  - Comm Manager(BCM) : used to select comm protocol and send status
    . BCM.c , BCM.h
- Lib(Common): contain common header files
  - Common Macros : comm_Macros.h
  - Std types : Std_types.h
- HAL :
  - Onboard layer :
    . Door sensor (D) : D.c , D.h
    . Light switch (L) : L.c , L.h
    . Speed sensor (S) : S.c , S.h
  - MCAL :
    . DIO : DIO.c , DIO.h
    . CAN : CAN.c , CAN.h
    . Timer : Timer.c , Timer.h

3. Provide full detailed APIs for each module as well as a detailed description for the used typedefs

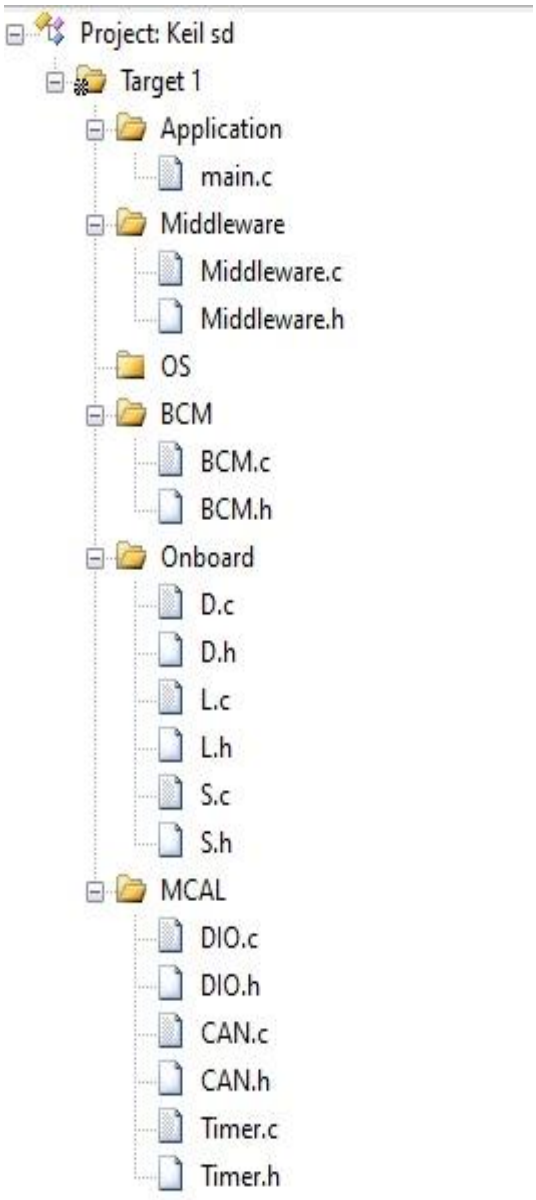
- Application module:
  - Void App_Init( );
    . initialize ECU , call all Peripheral modules (init functions)
    . Non-Reentrant, Synchronous, Non-Recursive Function.
  - Void App_start ( );
    . All program logic goes here.
    . Reentrant, Asynchronous, Non-Recursive Function
  - void Callback(void);
    .used with ISR to do some logic
    . Non-Reentrant, Synchronous, Non-Recursive Function
- Middleware module:
  - Void Middleware_sendData(Protocole_t protocole , data_t data) ;
    . select comm manager ,comm protocole and send data (status)
    . Reentrant, Asynchronous, Non-Recursive Function.
- BCM module:
  - BCM_CANsend(data_t data);
    . use CAN protocole to send data (status).
    . Reentrant, Asynchronous, Non-Recursive Function.
- Door Sensor(D) module:
  - D_Init(D_config_t* config);
    . take sensor configurations and initialize door sensor
    . Reentrant, Asynchronous, Non-Recursive Function.
  - D_state_t D_getState( );
    . Return door status (open || close)
    . Non-Reentrant, Synchronous, Non-Recursive Function
- Light switch (L) module:
  - L_Init(L_config_t* config);
    . take switch configurations and initialize Light switch
    . Reentrant, Asynchronous, Non-Recursive Function.
  - L_state_t L_getState( );
    . Return switch status (on || off)
    . Non-Reentrant, Synchronous, Non-Recursive Function
- Speed sensor (S) module:
  - S_Init(S_config_t* config);
    . take sensor configurations and initialize Speed sensor
    . Reentrant, Asynchronous, Non-Recursive Function.
  - S_value_t S_getValue( );
    . Return speed value
    . Non-Reentrant, Synchronous, Non-Recursive Function
- DIO module:
  - DIO_Init(DIO_config_t* config);
    . take DIO configurations and initialize DIO PORTs
    . Reentrant, Asynchronous, Non-Recursive Function.
  - DIO_level_t DIO_read(DIO_port_t port , DIO_pin_t pin);
    . Return pin value (LOW=0 || HIGH=1)
    . Non-Reentrant, Asynchronous, Non-Recursive Function.
  - DIO_write(DIO_port_t port , DIO_pin_t pin, Uint8_t value);
    . Set pin value ((LOW=0 || HIGH=1))
    . Non-Reentrant, Asynchronous, Non-Recursive Function.

- CAN module:
  - Void CAN_Init(CAN_config_t* config);
    . use CAN configurations to initialize CAN peripheral
    . Reentrant, Asynchronous, Non-Recursive Function
  - Void CAN_send(data_t data);
    . Tack data to be send
    . Non-Reentrant, Asynchronous, Non-Recursive Function
  - Data_t CAN_receive(data_t* data);
    . receive data
    . Reentrant, Synchronous, Non-Recursive Function
- Timer module:
  - Void Timer_Init(Timer_config_t config);
    . tack timer configurations to initialize timer peripheral
    . Reentrant, Asynchronous, Non-Recursive Function
  - Void Timer_start();
    . start timer after initialization
    . Non-Reentrant, Asynchronous, Non-Recursive Function.
  - Void timer_stop();
    . stop timer
    . Non-Reentrant, Asynchronous, Non-Recursive Function.
  - Void Timer_delay_ms(delay_t delat);
    . Delay time in ms
    . Non-Reentrant, Asynchronous, Non-Recursive Function.
- Common Macros module:
  - SET_BIT(Port, Pin)
  - CLEAR_BIT(Port, Pin)
  - READ_BIT(PORT,PIN)
    . Function-like macros to(set, clear , read)PINs

used typedefs

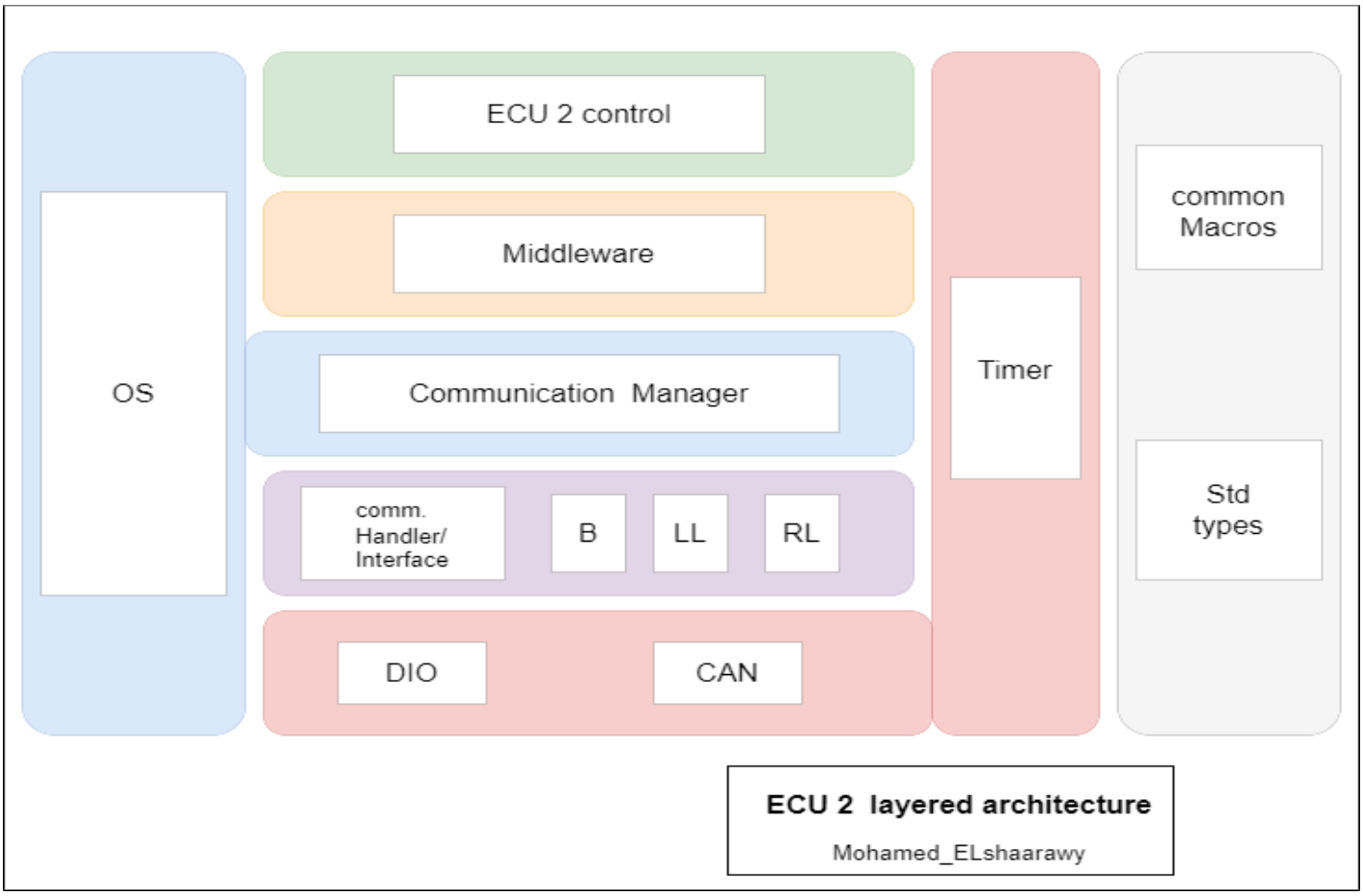
- Protocole_t : enum for protocol type (only CAN for this project)
- data_t : pre-defined type (uint8_t || uint16_t .....ext)
- D_config_t : struct for door sensor configuration
- D_state_t : enum for door status (open , close)
- L_config_t : struct for light switch configuration
- L_state_t : enum for light status (on , off)
- S_config_t : struct for speed sensor configuration
- S_value_t : pre-defined type (uint8_t || uint16_t .....ext)
- DIO_config_t : struct for DIO configuration
- DIO_level_t : enum for PIN level (LOW , HIGH)
- DIO_port_t : enum for PORT type (PORT_!,PORT_2,.....ext)
- DIO_pin_t : enum for PIN type (PIN_1 ,PIN_2.....ext)
- CAN_config_t : struct for CAN configuration
- Timer_config_t: struct for Timer configuration
- delay_t : pre-defined type (uint8_t || uint16_t .....ext)

4. Prepare your folder structure according to the previous points



For ECU 2:

1. Make the layered architecture



2. Specify ECU components and modules

- Application (ECU 2 control) : Where system logic and tasks are implemented.
  - Main.c
- Middleware : For routing the system to its desired destination.
  - Middleware.c , Middleware.h
- Services:
  - OS : assume (FreeRTOS)
    . all Kernel files
  - Comm Manager(BCM) : used to select comm protocol and send status
    . BCM.c , BCM.h
- Lib(Common): contain common header files
  - Common Macros : comm_Macros.h
  - Std types : Std_types.h
- HAL :
  - Onboard layer :
    . Right Light (RL) : RL.c , LL.h
    . Left Light (LL) : LL.c , LL.h
    . Buzzer (B) : B.c , B.h
  - MCAL :
    . DIO : DIO.c , DIO.h
    . CAN : CAN.c , CAN.h
    . Timer : Timer.c , Timer.h

3. Provide full detailed APIs for each module as well as a detailed description for the used typedefs

- Application module:
- Middleware module:
- BCM module:
- Right Light (RL) module:
  - Void RL_Init(RL_config_t* config);
    . take RL configurations and initialize RL
    . Reentrant, Asynchronous, Non-Recursive Function.
  - Void RL_on( );
    . turn RL ON
    . Non- Reentrant, Asynchronous, Non-Recursive Function.
  - Void RL_off( );
    . turn RL OFF
    . Non-Reentrant, Asynchronous, Non-Recursive Function.
  - RL_state_t RL_getState( );
    . Return RL status (ON || OFF)
    . Non-Reentrant, Synchronous, Non-Recursive Function
- Left light (LL) module:
  - Void RL_Init(LL_config_t* config);
    . take LL configurations and initialize LL
    . Reentrant, Asynchronous, Non-Recursive Function.
  - Void LL_on( );
    . turn LL ON
    . Non- Reentrant, Asynchronous, Non-Recursive Function.
  - Void LL_off( );
    . turn LL OFF
    . Non-Reentrant, Asynchronous, Non-Recursive Function.
  - LL_state_t LL_getState( );
    . Return LL status (ON || OFF)
    . Non-Reentrant, Synchronous, Non-Recursive Function

- Buzzer (B) module:
  - B_Init(B_config_t* config);
    . take Buzzer configurations and initialize Buzzer
    . Reentrant, Asynchronous, Non-Recursive Function.
  - Void B_on( );
    . turn Buzzer ON
    . Non- Reentrant, Asynchronous, Non-Recursive Function.
  - Void B_off( );
    . turn Buzzer OFF
    . Non-Reentrant, Asynchronous, Non-Recursive Function.

- DIO module:
- CAN module:
- Timer module:
- Common Macros module:
- Std types module

Unmentioned APIs are the same as ECU 1

- used typedefs
  - RL_config_t : struct for Right Light configuration
  - RL_state_t : enum for RL status (ON , OFF)
  - LL_config_t : struct for Left Light configuration
  - Ll_state_t : enum for LL status (ON , OFF)
  - B_config_t : struct for Buzzer configuration

4. Prepare your folder structure according to the previous points