중간 발표

# 목차

1. 목차 1

2. 프로젝트 개요 1

3. 데이터 Set 1

4. DataBase Model 7

5. AI System 8

6. FrontEnd, BackEnd 16

# 프로젝트 개요

## 우리 프로젝트팀은 프로젝트를, 본인의 문구점 아르바이트 경험으로 토대로 불편했던 점을 생각을 하면서 프로젝트 주제를 선정하였다.

## 문구점에서 아르바이트를 하면서 문구점에 수많은 물품의 위치를 찾기 힘들었고, 무분별하게 물건을 입고되는 경우가 있어서 물품이 방치되는 경우가 많았다.

## 이런 문제점을 해결하기 위해서 바로 Smart ERP System을 구현하게 되었다.

## 데이터를 처리하는 WEB Server, 월별 요일별로 물품 추천 해주는 AI Server, 물품의 위치를 찾아주는 Application을 구현하였으며, 상세적으로는 데이터를 처리하는 BackEnd Server는 Node.js Express 사용, 사용자에게 보여주는 FrontEnd는 EJS 플랫폼 사용, AI Server는 Keras Library를 사용한 LSTM(Long Short Term Memory) 모델 사용, Android 공기계로 Bluetooth Beacon를 활용한 물품 위치를 찾아주는 Application를 제작하였다.

# 데이터 Set

## AI 시스템을 돌리기 위해서는 언제나 데이터셋이 필요하다.

## 하지만 물품 입,출고량 데이터를 현직에서 장사하시는 분들에게도 구해보기로 했지만, 그 분들에게도 민감한 데이터라서 제공을 해주지 못한다고 하였다.

## 그래서 우리가 직접 데이터셋을 만드는 게 좋다고 판단을 하여, Python Language와, Pandas, Numpy, Random Library를 활용하여 DataSet을 제작하였다.

## Code는 다음과 같다.

|  |
| --- |
| from pandas import Series, DataFrameimport numpy as npimport datetimeimport random #1 민수, 2 다혜, 3 지수, 4 주완  name = np.array([1,2,3,4])  #1 색종이, 2 볼펜, 3 샤프, 4 A4용지, 5 찰흙, 6 코팅지  stationery = np.array([1,2,3,4,5,6])  sales\_data = {'입고날짜' : [], '요일' : [], '담당자' : [], '문구이름' : [], '입고가격' : [], '입고량' : [], '총입고가격' : []}  faulty\_data = {'입고날짜' : [], '요일' : [], '담당자' : [], '문구이름' : [], '불량개수' : []}  release\_data = {'출고날짜' : [], '요일' : [], '담당자' : [], '문구이름' : [], '출고가격' : [], '출고량' : [], '총출고가격' : []}  stock\_data = {'입고월' : [], '문구이름' : [], '재고' : []}  today = datetime.date(2000,1,1)  #stationery\_volume은 문구 수량  #stationery\_price는 문구 가격  stationery\_volume = {1 : 0, 2 : 0, 3 : 0, 4 : 0, 5 : 0, 6 : 0}  stationery\_price = {1 : 0, 2 : 0, 3 : 0, 4 : 0, 5 : 0, 6 : 0}  while True:  nowTuple = today.timetuple()  #년  year = nowTuple.tm\_year    #월  month = nowTuple.tm\_mon    #일  day = nowTuple.tm\_mday    #요일  weekday = nowTuple.tm\_wday  #년도가 2019년이라면 break  if(year == 2020):  break  #만약 요일이 토요일 또는 일요일 경우  elif (weekday == 6 or weekday == 5):  today = today + datetime.timedelta(days=1)  else:  #입고 <- 하루에 입고량은 랜덤으로 생성 (물량이 없을 시 입고 되는 거라서 상관 없을 듯)  rand\_day\_sales = random.randrange(1,40)    for i in range(1,rand\_day\_sales):  #name\_value는 담당자  name\_value = np.random.choice(name)  #stationery\_value는 문구류  stationery\_value = np.random.choice(stationery)  #volume는 문구류의 값을 가지고 온 것  volume = stationery\_volume.get(stationery\_value)    #입고량이 0일 경우 입고 하기  if(volume <=20):  volume = random.randrange(20,100)  price = random.randrange(1000,300000)  sales\_data['입고날짜'].append(today)  sales\_data['요일'].append(weekday)  sales\_data['담당자'].append(name\_value)  sales\_data['문구이름'].append(stationery\_value)  sales\_data['입고가격'].append(price)  sales\_data['입고량'].append(volume)  all\_price = price \* volume  sales\_data['총입고가격'].append(all\_price)  stationery\_price[stationery\_value] = price    #불량  volume\_temp = int(volume / 5)    if(volume\_temp == 0):  volume\_temp = 1    faulty\_temp = random.randrange(0, volume\_temp)    if(faulty\_temp == 0):  continue  else:  faulty = volume - faulty\_temp  stationery\_volume[stationery\_value] = faulty  stationery\_price[stationery\_value] = price  faulty\_data['입고날짜'].append(today)  faulty\_data['요일'].append(weekday)  faulty\_data['담당자'].append(name\_value)  faulty\_data['문구이름'].append(stationery\_value)  faulty\_data['불량개수'].append(faulty\_temp)    rand\_day\_release = random.randrange(1,100)    #출고  for i in range(1,rand\_day\_release):  #name\_value는 담당자  name\_value = np.random.choice(name)  #stationery\_value는 문구류  stationery\_value = np.random.choice(stationery)  #volume는 문구류의 값을 가지고 온 것  volume = stationery\_volume.get(stationery\_value)    if(volume <= 1):  continue    #가격  price = stationery\_price.get(stationery\_value)  #출고값 랜덤  random\_value = random.randrange(1,volume)  #입고 - 출고값  volume\_result = volume - random\_value  price = price + (price \* 0.1)  all\_price = price \* volume\_result    release\_data['출고날짜'].append(today)  release\_data['요일'].append(weekday)  release\_data['담당자'].append(name\_value)  release\_data['문구이름'].append(stationery\_value)  release\_data['출고가격'].append(price)  release\_data['출고량'].append(random\_value)  release\_data['총출고가격'].append(all\_price)    stationery\_volume[stationery\_value] = volume\_result      #재고  stock\_today = str(year) + "-" + str(month)    for i in range(1,6):  volume = stationery\_volume.get(i)  #31일 경우  if(day==31):  stock\_data['입고월'].append(stock\_today)  stock\_data['문구이름'].append(i)  stock\_data['재고'].append(volume)  continue    if(day==30):  stock\_data['입고월'].append(stock\_today)  stock\_data['문구이름'].append(i)  stock\_data['재고'].append(volume)  continue    if(month == 2 and day == 29 or day ==28):  stock\_data['입고월'].append(stock\_today)  stock\_data['문구이름'].append(i)  stock\_data['재고'].append(volume)  continue    today = today + datetime.timedelta(days=1)  #CSV 파일 출력  sales\_frame = DataFrame(sales\_data)  sales\_frame.index +=1  sales\_frame.to\_csv('sales.csv',mode='w',encoding='ms949')  faulty\_frame = DataFrame(faulty\_data)  faulty\_frame.index +=1  faulty\_frame.to\_csv('faulty.csv',mode='w',encoding='ms949')  stock\_frame = DataFrame(stock\_data)  stock\_frame.index +=1  stock\_frame.to\_csv('stock.csv',mode='w',encoding='ms949')  release\_frame = DataFrame(release\_data)  release\_frame.index +=1  release\_frame.to\_csv('release.csv', mode='w', encoding='ms949') |

## 결과는 다음과 같다.

## 

## 다음과 같이 데이터셋이 생성이 되었다.

## 데이터셋을 얻을 수가 없어서 막막했지만 코드로 데이터셋을 임의적으로 만들었으니 천만 다행이라고 생각이 든다.

# DataBase Model

## DataBase는 MySQL를 사용하여 DataBase를 구축하였다.

## 처음에 구축한 DataBase 모델 초안이 있었지만, 개발을 진행을 하던 도중, 필요한 Table이 생기기 시작하여, 지금의 DataBase처럼 모델링이 완성되었다.

## 

## 총 9개의 Table으로 입고, 재고, 출고, 출고대기, 사용자, 물품, 학습결과, 게시판, 불량 게시판으로 구성하였다.

## 관계는 BCNF 정규형으로 분해하여 Join을 하였으며, BCNF 정규형이란 후보키가 아닌 결정자를 제거해주는 것이므로, 즉 일부 데이터가 갱신, 삽입, 삭제가 되어도 이상이 일어나지 않게 분해를 하였다.

# AI System

## AI System은 앞서 언급했던 Keras의 LSTM 모델을 사용하여 구현하였다.

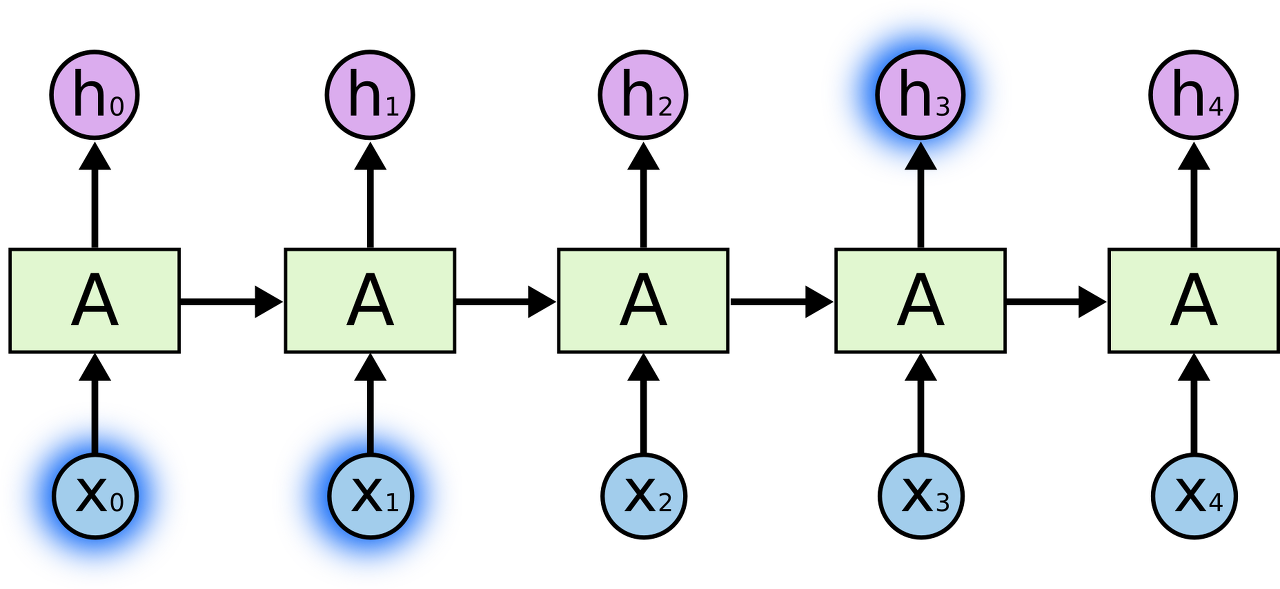
## LSTM은 Long Short Term Memory 의 약자로써 RNN (Recurrent Neural Network)를 한 단계 더 발전한 모델이다.

## RNN의 개념부터 설명하자면 자기 스스로 반복하면서 이전 단계에서 얻은 정보를 지속되도록 하는 것이다. 예를 들어 우리가 하루에 일어나는 일들을 분류하고 싶다고 해보자.

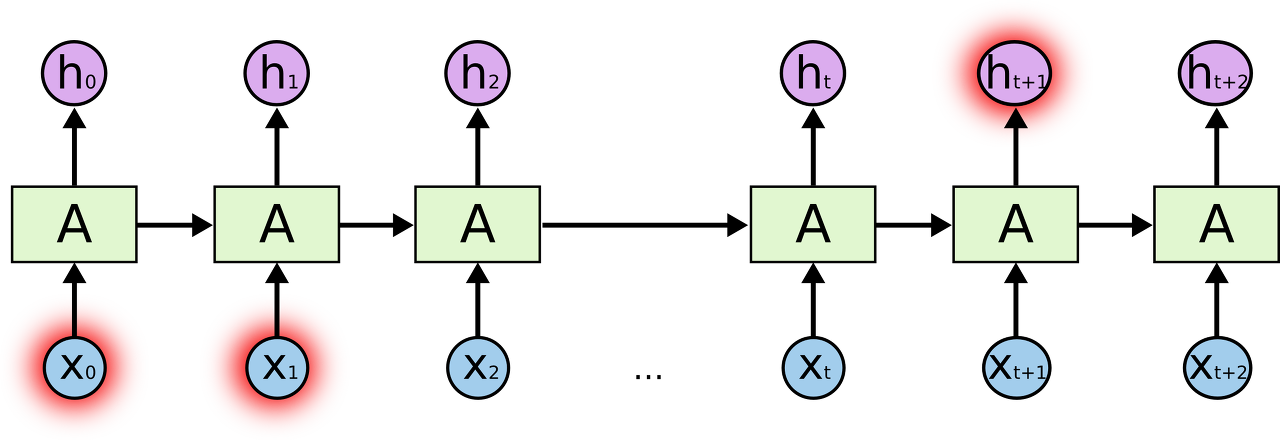
## 전통적인 Neural Network는 이전에 일어난 사건을 바탕으로 나중에 일어나는 사건을 생각하지 못한다 즉 미래를 생각을 못한다고 볼 수 있다.

## 이러한 문제점을 해결하기 위해서 RNN이 모델이 개발 되었고, 음성 인식, 언어 모델링, 번역, 이미지 주석 생성 등등 다양한 분야에서 사용되고 있다.

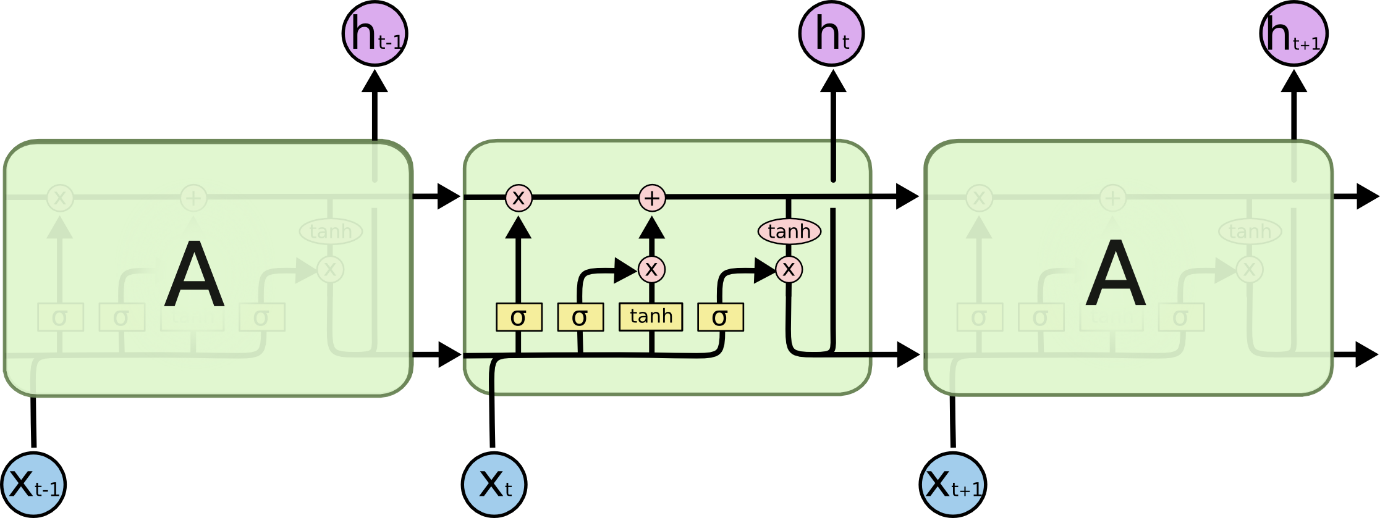
## 하지만 RNN의 문제점은 긴 의존기간으로 인해 문제점이 생기는데, 긴 의존기간의 문제점이란 기간이 긴 데이터를 예측 할 시 시간격차가 굉장히 커지게 된다고 한다.



짧은 시간의 RNN 작동 방식



긴 시간의 RNN의 구현 방식



LSTM 구현 방식

## 그래서 이런 긴 기간의 데이터를 예측 할 때 시간 격차를 좁히기 위해서 개발된 모델이 LSTM 모델이라고 한다.

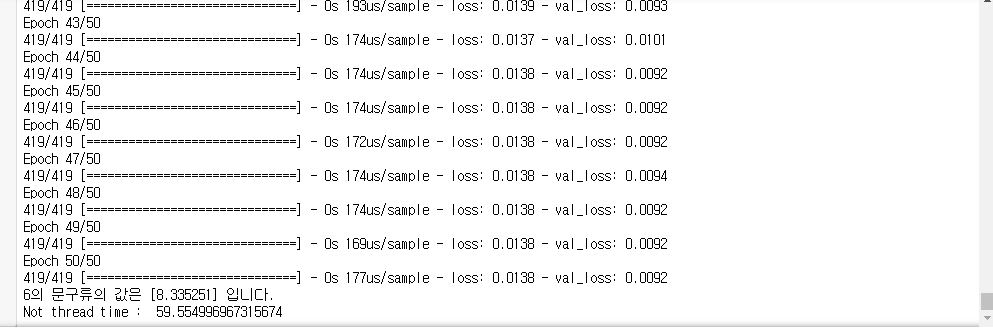
## 우리는 오래된 데이터까지 학습하여 결과를 추출을 해야 되기 때문에, LSTM 모델을 선택을 하였고, 추출한 데이터를 학습을 시켜보았다.

## 데이터 학습을 시킨 컴퓨터의 성능은 CPU는 I5-6500, GPU GTX650, RAM 8G의 성능을 가진 PC이다.

## 학습을 해본 결과 속도는 약 60초 정도 나왔다. RNN보다는 만족 할 수는 있겠지만, 한 번 더 생각을 해보기로 했다. ‘물품당 Thread를 생성해서 학습을 시키면 더 빠르지 않을까?’라는 생각을 했고 물품당 Thread를 생성하여 학습을 해본 결과 약 50초 정도의 매우 빠른 속도로 학습이 되었다.

## 

## Thread로 학습 시 약 47초 시간이 소요되었다. (요일마다 데이터셋이 다르니 약 50초 생각하면 될 듯 하다.)



## Thread를 미 사용 학습 시 약 59초 시간이 소요되었다. (요일마다 데이터셋이 다르니 약 60초 생각하면 될 듯하다.)

## 학습 된 데이터의 결과를 DB에 INSERT 하면 다음과 같이 데이터가 INSERT가 된 걸 확인 할 수 있다.

## 

## 학습 결과 데이터 DB에 삽입

## Code는 다음과 같다.

|  |
| --- |
| import pymysql as db  import numpy as np  import pandas as pd  import datetime  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import LSTM, Dropout, Dense, Activation  from sklearn.preprocessing import MinMaxScaler  import threading as t  import time  from sqlalchemy import create\_engine  db.install\_as\_MySQLdb()  import MySQLdb  #출고 데이터 저장 딕셔너리  df = 0  #현재 날짜 가지고 오기  now = datetime.datetime.now()  #현재 날짜를 년,월,요일 등으로 분리  now\_result = now.timetuple()  #요일  weekday = now\_result.tm\_wday  #월  month = now\_result.tm\_mon  #문구류\_List  Stationery\_list = []  #학습결과  result\_dict = {"Stationery\_NO" : [], "Count" : [], "Today" : []}  result\_df = 0  class DB\_Connect:  conn = db.connect(host='localhost', user='root', password='1234', db='ai\_db', charset='utf8')  curs = conn.cursor(db.cursors.DictCursor)  #DB\_SQL  class DB\_SQL(DB\_Connect):  #DB값 읽기  def db\_read(self,sql):  print(sql)  DB\_Connect.curs.execute(sql)  print("db\_read 성공!")  result = DB\_Connect.curs.fetchall()  df = pd.DataFrame(result)  #문구류의 값 리스트에 넣어서 중복 제거  Stationery\_list = set(df['Stationery\_NO'])  DB\_Connect.conn.close()    return df, Stationery\_list    #DB값 쓰기  def db\_write(self,data):  engine = create\_engine("mysql+mysqldb://root:" + "1234" + "@localhost:3306/ai\_db", encoding='utf-8')  conn = engine.connect()  data  data.to\_sql(name = "training\_results", con = engine, if\_exists='append')  print('테이블 데이터 업데이트 완료!')  #DB\_Data\_Read  def data\_read():  db\_result = DB\_SQL()    if(weekday == 6 or weekday == 5):  print("금일을 휴일입니다.")  else:  sql = "select Stationery\_NO, Count, MONTH(Date) as 'months' from ai\_db.releases where Weekday = " + str(weekday) + " and MONTH(Date) = " + str(month) + ";"    global df  global Stationery\_list    df,Stationery\_list = db\_result.db\_read(sql)  #학습  class AI\_Model():  #데이터셋 변환 과정  def create\_dataset(data, look\_back=1):  x\_arr, y\_arr = [], []    for i in range(len(data) - look\_back):  x\_arr.append(data[i:(i + look\_back),0])  y\_arr.append(data[i + look\_back, 0])    x\_arr = np.array(x\_arr)  x\_arr = np.reshape(x\_arr, (x\_arr.shape[0], x\_arr.shape[1],1))    return x\_arr, np.array(y\_arr)    #학습  def training(no):  #데이터 전처리 (0 ~ 1 범위)  scaler = MinMaxScaler(feature\_range=(0,1))  #지금 전체 카운트롤 계산 했지만 Thread 문구류 각각해서 동시에 돌릴 예정  data\_temp = (df['Stationery\_NO'] == no)  data\_result = df[data\_temp]  print(data\_result)    temp = df[data\_temp].values.reshape(-1,1)  trade\_count = scaler.fit\_transform(temp)    #훈련  train = trade\_count[0:int(len(trade\_count) \* 0.5)]  #검증  val = trade\_count[int(len(trade\_count) \* 0.5) : int(len(trade\_count) \*0.75)]  #시험  test = trade\_count[int(len(trade\_count) \* 0.75) : -1]    #x,y 트레이닝 셋  x\_train, y\_train = AI\_Model.create\_dataset(train, 1)  #x,y 검증셋  x\_val, y\_val = AI\_Model.create\_dataset(val,1)  #x,y 시험 셋  x\_test, y\_text = AI\_Model.create\_dataset(val,1)    #학습 모델 구성  batch\_size = 22  model = Sequential()    model.add(LSTM(50, return\_sequences=True, input\_shape=(1,1)))  model.add(LSTM(64,return\_sequences=False))  model.add(Dense(1, activation='linear'))    #손실  model.compile(loss='mse', optimizer='rmsprop')  model.summary()  model.fit(x\_train, y\_train, validation\_data=(x\_val, y\_val), batch\_size=batch\_size, epochs=50)  predictions = model.predict(x\_test, batch\_size)  read\_predictions = scaler.inverse\_transform(predictions)  result = read\_predictions[-1]    print(str(no) + "의 문구류의 값은 " + str(result) +" 입니다.")    #학습결과\_Dict  global result\_dict  result\_dict['Stationery\_NO'].append(no)  result\_dict['Count'].append(int(result))  result\_now = str(now\_result.tm\_year) + "-" + str(now\_result.tm\_mon) + "-" + str(now\_result.tm\_mday)  result\_dict['Today'].append(result\_now)  #쓰레드 생성, 실행 함수  def run\_threads(list\_temp):  threads = []    for i in range(len(list\_temp)):  thread = t.Thread(target=AI\_Model.training,kwargs={"no" : list\_temp[i]})  threads.append(thread)  thread.start()    for thread in threads:  thread.join()  data\_read()  #모델 객체 생성  ai\_model = AI\_Model()  list\_temp = list(Stationery\_list)  start = time.time()  #학습 모델 쓰레드 생성  run\_threads(list\_temp)  #쓰레드 미생성  # for i in range(len(list\_temp)):  # ai\_model.training(list\_temp[i])  print('thread time : ', time.time() - start)  result\_df = pd.DataFrame(result\_dict)  result\_df.index +=1  result\_df.index.names = ['id']  result\_df  db\_result = DB\_SQL()  db\_result.db\_write(result\_df) |

## 만족 할 만한 학습 결과가 나와서 매우 만족한다.

# FrontEnd, BackEnd

## FrontEnd와 BackEnd는 전임 개발자가 있었지만, 개발을 다 마치지 못하고 군대로 가 버리고 말았다.

## 그래서 전임자가 했던 걸 인수인계를 받아서 개발을 시작을 하였지만, 전임자가 인수인계서 만들지도 않고, 코드에 주석도 달지 않고 군대로 입대해 버린 것이다.

## 그래서 처음부터 개발을 할까 고민을 하다가, 전임자가 했던 걸 하나 하나씩 해석을 하는데 시간이 엄청 많이 소요가 되었다.

## 해석을 다 한 후에, 개발을 시작하였는데, 빼야 될 것도 많고, 코드 정리도 해야 될 게 많아서 아직 개발 단계에 있다.

## 현재는 코드 최적화, DB 연동, 로그인을 구현하였으며, DB 연동은 Node.js에서 쉽게 DB를 관리할 수 있는 라이브러리 Sequelize를 사용 하였고, 로그인은 세션까지 쉽게 유지 할 수 있는 PassPort 라이브러리를 사용하였다.

## 

## Sequelize 라이브러리는 Node.js 에서 SQL문을 모델링을 하여 DB를 관리하며, 기존에 JQuery를 활용한 SQL문을 지저분해서 코드의 가독성이 떨어졌다면, Sequelize는 DB를 객체로 모델링해서 관리하기 때문에 코드의 가독성이 매우 좋고 DB를 쉽게 관리 할 수 있다.

## 

## PassPort 여권이라는 뜻을 가지며 Node.js에서 기존에 사용자가 로그인 시 세션이나 각종 사용자의 값들을 따로따로 관리를 해야 되기 때문에, 불편하였는데, Passport를 활용하면 사용자의 값들과 세션들을 한 번에 관리를 할 수 있기 때문에 개발자들은 매우 편리하게 개발을 할 수 있는 라이브러리이다.

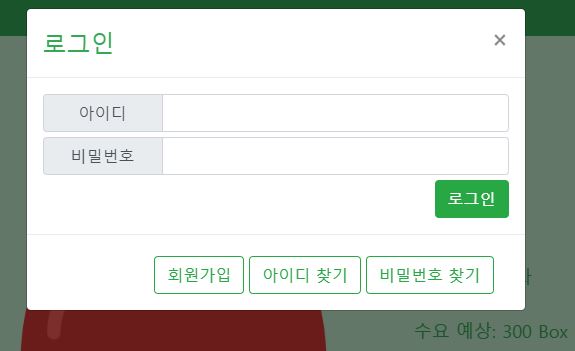
## 일단 Web Project 폴더 구성은 다음과 같다.

## 

## 이렇게 구성되어 있으며, 핵심 폴더만 설명하자면, models는 DB 모델링 한 폴더, passport는 passport 라이브러리를 활용해서 로그인, 로그아웃, 회원가입 구현, views는 사용자에게 보여주는 FrontEnd 부분, Routes는 서버가 동작하는 BackEnd 부분이다.

## 

## Main 화면이다. 전임자가 했던 걸 제대로 수정하지 않은 상태라 내용들이 많이 빠진 게 보인다.



## 로그인 창이다. 지금 아이디만 입력하면 로그인이 가능한 상태이다. 비밀번호로 로그인도 해야 되지만 회원가입부분을 아직 구현이 되지 않아서 회원가입부분 구현 후 구현 할 것이다.

나눠서 한 이유는 바로 Hash 암호화 알고리즘을 사용해서 비밀번호를 암호화를 할 생각이기 때문이다.



로그인이 완료되면 다음과 같이 로그인 버튼이 사라지고 사용자 정보, 로그아웃, 회원정보 변경 창이 뜬다.

아직 미흡한 부분이 많지만 최대한 많이 찾아가면서 Web 개발을 마무리해야 될 것 같다.