application.conf
==========

```
[LOCAL]
customers.file.path = data/customers.csv
orders.file.path = data/orders.csv

[TEST]
customers.file.path = data/customers.csv
orders.file.path = data/orders.csv

[PROD]
customers.file.path = data/customers.csv
orders.file.path = data/orders.csv
```

pyspark.conf
==========

```
[LOCAL]
spark.app.name = retail-local

[TEST]
spark.app.name = retail-test
spark.executor.instances = 3
spark.executor.cores = 5
spark.executor.memory = 15GB

[PROD]
spark.app.name = retail-prod
spark.executor.instances = 3
spark.executor.cores = 5
spark.executor.memory = 15GB
```

ConfigReader.py
==========

```
import configparser
from pyspark import SparkConf

# loading the application configs in python dictionary
def get_app_config(env):
    config = configparser.ConfigParser()
    config.read("configs/application.conf")
    app_conf = {}
```

```
    for (key, val) in config.items(env):
        app_conf[key] = val
    return app_conf

# loading the pyspark configs and creating a spark conf object
def get_pyspark_config(env):
    config = configparser.ConfigParser()
    config.read("configs/pyspark.conf")
    pyspark_conf = SparkConf()
    for (key, val) in config.items(env):
        pyspark_conf.set(key, val)
    return pyspark_conf
```

DataManipulation.py
==========

```
from pyspark.sql.functions import *

def filter_closed_orders(orders_df):
    return orders_df.filter("order_status = 'CLOSED'")

def join_orders_customers(orders_df, customers_df):
    return orders_df.join(customers_df, "customer_id")

def count_orders_state(joined_df):
    return joined_df.groupBy('state').count()
```

DataReader.py
==========

```
from lib import ConfigReader

#defining customers schema
def get_customers_schema():
    schema = "customer_id int,customer_fname string,customer_lname
string,username string,password string,address string,city string,state
string,pincode string"
    return schema

# creating customers dataframe
def read_customers(spark,env):
    conf = ConfigReader.get_app_config(env)
    customers_file_path = conf["customers.file.path"]
    return spark.read \
        .format("csv") \
```

```
        .option("header", "true") \
        .schema(get_customers_schema()) \
        .load(customers_file_path)

#defining orders schema
def get_orders_schema():
    schema = "order_id int,order_date string,customer_id int,order_status
string"
    return schema

#creating orders dataframe
def read_orders(spark,env):
    conf = ConfigReader.get_app_config(env)
    orders_file_path = conf["orders.file.path"]
    return spark.read \
        .format("csv") \
        .option("header", "true") \
        .schema(get_orders_schema()) \
        .load(orders_file_path)
```

Utils.py
=========

```
from pyspark.sql import SparkSession
from lib.ConfigReader import get_spark_conf

def get_spark_session(env):
    if env == "LOCAL":
        return SparkSession.builder \
            .config(conf=get_spark_conf(env)) \
            .master("local[2]") \
            .getOrCreate()
    else:
        return SparkSession.builder \
            .config(conf=get_spark_conf(env)) \
            .enableHiveSupport() \
            .getOrCreate()
```

application_main.py
==========

```
import sys
from lib import DataManipulation, DataReader, Utils
from pyspark.sql.functions import *

if __name__ == '__main__':

    if len(sys.argv) < 2:
        print("Please specify the environment")
        sys.exit(-1)

    job_run_env = sys.argv[1]

    print("Creating Spark Session")

    spark = Utils.get_spark_session(job_run_env)

    print("Created Spark Session")

    orders_df = DataReader.read_orders(spark,job_run_env)

    orders_filtered = DataManipulation.filter_closed_orders(orders_df)

    customers_df = DataReader.read_customers(spark,job_run_env)

    joined_df =
DataManipulation.join_orders_customers(orders_filtered,customers_df)

    aggregated_results = DataManipulation.count_orders_state(joined_df)

    aggregated_results.show()

    print("end of main")
```