4. IMPLEMENTASI SISTEM

Pada bab ini akan menjelaskan mengenai modul apa saja yang dikustomisasi sesuai dengan kebutuhan Toko Best Charcoals. Konfigurasi modul dilakukan pada modul *sales* dan *inventory*. Untuk efisiensi operasional Toko Best Charcoals, dilakukan pembuatan modul *sales forecast* dan kustomisasi pada *replenishment*.

4.1. Implementasi Program

Implementasi sistem menerapkan desain sistem yang terdapat di Bab 3. Implementasi dapat di lihat pada Tabel 4.1.

Tabel 4.1 Menu dan Proses Kegiatan Pada Sistem

Nama Use Case	Nama Submenu	Keterangan	Segmen Program
Sales Forecast	Forecast List	Menampilkan daftar *forecast* yang telah dilakukan	4.1 dan 4.2
Inventory	Replenishment	Mengubah dan memesan stok produk kepada supplier untuk pengisian ulang.	4.3 dan 4.4

4.2. Proses *Sales Forecast*

4.2.1 Implementasi metode ARIMA

Setelah dilakukan ekstraksi untuk pengelompokkan tanggal berdasarkan bulan dan nomor barang pada *dataframe*, selanjutnya dilakukan tes untuk menentukan *stationarity* menggunakan *Augmented Dickey-Fuller*. Setelah diketahui bahwa data tersebut stasioner atau tidak, lalu dilanjutkan dengan *fitting* model yang ditunjukkan pada segmen program 4.1 untuk menghasilkan angka *forecast* dalam periode yang telah ditentukan oleh user berdasarkan metode ARIMA.

Segmen Program 4.1 *Forecast* ARIMA

```
pattern = r'PJ-(\d{4}-\d{2}-\d{2})-\d{4}'
df['extracted_date'] = df['extracted_date'].str.extract(pattern)
df['extracted_date'] = pd.to_datetime(df['extracted_date'], format='%Y-%m-%d')
df = df[['extracted_date', 'Jumlah_penjualan', 'Kode_barang']]
df['Kode_barang'] = df['Kode_barang'].astype(int)
```

```
def calculate_and_merge_sales(chosen_kode_barang):
   filtered_df = df[df['Kode_barang'] == chosen_kode_barang]
   filtered_df.set_index('extracted_date', inplace=True)
   monthly_sales = filtered_df.resample('M')['Jumlah_penjualan'].sum()

   df_sales = df[df['Kode_barang'] == chosen_kode_barang][['Jumlah_penjualan',
'extracted_date']]
   df_sales.set_index('extracted_date', inplace=True)

   merged_df = pd.merge(filtered_df, df_sales, left_index=True, right_index=True, how='left')
   merged_df.rename(columns={'Jumlah_penjualan_x': 'Jumlah_penjualan', 'Jumlah_penjualan_y':
'Monthly_Sales'},
                inplace=True)
   merged_df.reset_index(inplace=True)
   merged_df['Month'] = merged_df['extracted_date'].dt.to_period('M')

   return merged_df

chosen_kode_barang = int(self.product_default_code)

df_with_sales = calculate_and_merge_sales(chosen_kode_barang)
df_with_sales.reset_index(inplace=True)
df_with_sales.set_index('Month', inplace=True)
sum_sales_by_month = df_with_sales.groupby('Month').agg({'Monthly_Sales': 'sum'})

# use this to escape error like "float() argument must be a string or a number, not 'Period'"
pd.plotting.register_matplotlib_converters()

sum_sales_by_month["Month"] = sum_sales_by_month.index.to_timestamp()
sum_sales_by_month.index = sum_sales_by_month.index.to_timestamp()

# Determine rolling statistics
sum_sales_by_month["rolling_avg"] =
sum_sales_by_month["Monthly_Sales"].rolling(window=12).mean()
sum_sales_by_month["rolling_std"] =
sum_sales_by_month["Monthly_Sales"].rolling(window=12).std()

# Augmented Dickey–Fuller test:
dftest = adfuller(sum_sales_by_month['Monthly_Sales'], autolag='AIC')

dfoutput = pd.Series(dftest[0:4],
                index=['Test Statistic', 'p-value', '#Lags Used',
                     'Number of Observations Used'])
for key, value in dftest[4].items():
   dfoutput['Critical Value (%s)' % key] = value

def obtain_adf_kpss_results(timeseries, max_d):
   results = []
```

```
    for idx in range(max_d):
        adf_result = adfuller(timeseries, autolag='AIC')
        kpss_result = kpss(timeseries, regression='c', nlags="auto")
        timeseries = timeseries.diff().dropna()
        if adf_result[1] <= 0.05:
            adf_stationary = True
        else:
            adf_stationary = False
        if kpss_result[1] <= 0.05:
            kpss_stationary = False
        else:
            kpss_stationary = True

        stationary = adf_stationary & kpss_stationary

        results.append(
            (idx, adf_result[1], kpss_result[1], adf_stationary, kpss_stationary, stationary))

    # Construct DataFrame
    results_df = pd.DataFrame(results, columns=['d', 'adf_stats', 'p-value', 'is_adf_stationary',
                                                'is_kpss_stationary', 'is_stationary'])
    return results_df

new_period = self.forecast_duration
new_period *= 12

def calculate_mape(actual, forecast):
    actual = np.array(actual)
    forecast = np.array(forecast)

    if actual.shape != forecast.shape:
        raise ValueError("Actual and forecast arrays must have the same shape.")
    # Calculate absolute percentage error for each data point
    abs_percentage_error = np.abs((actual - forecast) / actual)
    # Calculate the mean absolute percentage error
    mape = np.mean(abs_percentage_error) * 100.0

    return mape

# Initialize variables to track the best MAPE and order
best_mape = np.inf
best_order = None

for p in range(1, 11):
    for d in range(1):
        for q in range(1, 11):
            try:
                ARIMA_model = ARIMA(sum_sales_by_month['Monthly_Sales'], order=(p, d, q))
                ARIMA_results = ARIMA_model.fit()
```

```
            new_forecasts = ARIMA_results.fittedvalues
            new_forecasts[0] = sum_sales_by_month['Monthly_Sales'].iloc[0]

            mape = calculate_mape(sum_sales_by_month['Monthly_Sales'].values, new_forecasts)

            # Check if this MAPE is better than the best so far
            if mape < best_mape:
                best_mape = mape
                best_order = (p, d, q)
        except:
            continue
```

4.2.2 Implementasi metode Prophet

Setelah dilakukan ekstraksi untuk pengelompokkan tanggal berdasarkan bulan dan nomor barang pada *dataframe*, selanjutnya dilakukan tes untuk menentukan *stationarity* menggunakan *Augmented Dickey-Fuller*. Setelah diketahui bahwa data tersebut stasioner atau tidak, lalu dilanjutkan dengan *fitting* model yang ditunjukkan pada segmen program 4.2 untuk menghasilkan angka *forecast* dalam periode yang telah ditentukan oleh user berdasarkan metode Prophet.

Segmen Program 4.2 *Forecast* Prophet

```
pattern = r'PJ-(\d{4}-\d{2}-\d{2})-\d{4}'
df['extracted_date'] = df['extracted_date'].str.extract(pattern)
df['extracted_date'] = pd.to_datetime(df['extracted_date'], format='%Y-%m-%d')
df['Kode_barang'] = df['Kode_barang'].astype(int)
df = df[['extracted_date', 'Jumlah_penjualan', 'Kode_barang']]

# sales per month
def calculate_and_merge_sales(chosen_kode_barang):
    filtered_df = df[df['Kode_barang'] == chosen_kode_barang]
    filtered_df.set_index('extracted_date', inplace=True)
    monthly_sales = filtered_df.resample('M')['Jumlah_penjualan'].sum()

    df_sales = df[df['Kode_barang'] == chosen_kode_barang][['Jumlah_penjualan',
'extracted_date']]
    df_sales.set_index('extracted_date', inplace=True)

    merged_df = pd.merge(filtered_df, df_sales, left_index=True, right_index=True, how='left')
    merged_df.rename(
        columns={'Jumlah_penjualan_x': 'Jumlah_penjualan', 'Jumlah_penjualan_y':
'Monthly_Sales'},
        inplace=True)
```

```
    merged_df['Month'] = merged_df.index.to_period('M')

    return merged_df

# sales per day
def calculate_and_merge_sales_per_day(chosen_kode_barang):
    filtered_df = df[df['Kode_barang'] == chosen_kode_barang]

    df_sales = df[df['Kode_barang'] == chosen_kode_barang][['Jumlah_penjualan',
'extracted_date']]

    merged_df = pd.merge(filtered_df, df_sales, on='extracted_date', how='left')
    merged_df.rename(
        columns={'Jumlah_penjualan_x': 'Jumlah_penjualan', 'Jumlah_penjualan_y':
'Sales_Per_Date'},
        inplace=True)

    return merged_df

chosen_kode_barang = int(self.product_default_code)

df_with_sales = calculate_and_merge_sales(chosen_kode_barang)
df_with_sales.set_index('Month', inplace=True)
sum_sales_by_month = df_with_sales.groupby('Month').agg({'Monthly_Sales': 'sum'})
sum_sales_by_month['month'] = sum_sales_by_month.index.to_timestamp()
y = sum_sales_by_month
new_df = calculate_and_merge_sales_per_day(chosen_kode_barang)
x = new_df[['extracted_date', 'Jumlah_penjualan']]
y['month'] = pd.DatetimeIndex(y['month'])
y = y[['month', 'Monthly_Sales']]
df = y.rename(columns={'month': 'ds', 'Monthly_Sales': 'y'})

# Determine rolling statistics
y["rolling_avg"] = y["Monthly_Sales"].rolling(
    window=12).mean()  # window size 12 denotes 12 months, giving rolling mean at yearly level
y["rolling_std"] = y["Monthly_Sales"].rolling(window=12).std()

# Augmented Dickey–Fuller test:
# print('Results of Dickey Fuller Test:')
dftest = adfuller(y['Monthly_Sales'], autolag='AIC')

dfoutput = pd.Series(dftest[0:4],
                     index=['Test Statistic', 'p-value', '#Lags Used', 'Number of Observations Used'])
for key, value in dftest[4].items():
    dfoutput['Critical Value (%s)' % key] = value

# print(dfoutput)

df.columns = ['ds', 'y']
```

```
p = Prophet(interval_width=0.95)  # interval width default is 80%
p.add_country_holidays(country_name='ID')
p.fit(df)
p.train_holiday_names

new_period = self.forecast_duration
new_period *= 12

future = p.make_future_dataframe(periods=new_period, freq='MS')
forecast_prediction = p.predict(future)
forecast_prediction['yhat'] = forecast_prediction['yhat'].round(0)
plot1 = p.plot(forecast_prediction, uncertainty=True)
# Save the plot as an image
image_stream = io.BytesIO()
plot1.savefig(image_stream, format='png')

# Store the plot image in the field
self.plot_image_prophet = base64.b64encode(image_stream.getvalue()).decode('utf-8')

# plot1 = p.plot(forecast_prediction)
# a = add_changepoints_to_plot(plot1.gca(),p,forecast_prediction)
plot2 = p.plot_components(forecast_prediction)
# Save the plot as an image
image_stream2 = io.BytesIO()
plot2.savefig(image_stream2, format='png')

# Store the plot image in the field
self.plot_image_prophet2 = base64.b64encode(image_stream2.getvalue()).decode('utf-8')

df['ds'] = pd.to_datetime(df['ds'])
# Combine the relevant columns from x and forecast_prediction DataFrames
df_merge = pd.merge(df, forecast_prediction[['ds', 'yhat_lower', 'yhat_upper', 'yhat']], on='ds')
df_merge = df_merge[['ds', 'yhat_lower', 'yhat_upper', 'yhat', 'y']]

# calculate MAPE between observed and predicted values
y_true = df_merge['y'].values
y_pred = df_merge['yhat'].values
mape_01 = mean_absolute_percentage_error(y_true, y_pred)
formatted_mape = '{:.2%}'.format(mape_01 / 1)
for i in self:
   i.update({
      'mape_prophet_output': formatted_mape,
   })

sum_sales_by_month['Month'] = sum_sales_by_month.index.to_timestamp()
sum_sales_by_month.index = sum_sales_by_month.index.to_timestamp()
last_date_prophet = sum_sales_by_month.index.max()
index_for_prophet = pd.date_range(start=last_date_prophet + pd.DateOffset(months=1),
                        periods=new_period, freq='MS')
```

```
y_num = round(forecast_prediction['yhat'], 0)
```

4.3. Proses Bisnis Pembelian

4.3.1 Penentuan Safety Stock

Untuk menentukan angka *safety stock* dan *maximum quantity* akan bergantung ketika ada perubahan angka pada kolom *'product_id'*, *'qty_forecast'*, *'product_min_qty'*, dan *'product_max_qty'*.

Segmen Program 4.3 Penentuan angka *Safety Stock* dan *Maximum Quantity*

```
@api.depends('product_id', 'qty_forecast', 'product_min_qty', 'product_max_qty')
  def _get_min_max_qty(self):
    config_settings = self.env['res.config.settings'].sudo().create({})

    for rec in self:
      fore_product_ids = self.env['sale.forecast'].search([
        ('product_id', '=', rec.product_id.id)
      ])

      for fore_product_id in fore_product_ids:
        sale_forecast_lines = self.env['sale.forecast.line'].search([
          ('forecast_id', '=', fore_product_id.product_default_code)
        ])

        x = []

        for sale_forecast_line in sale_forecast_lines:
          # Assuming forecasted_unit is a float
          x.append(sale_forecast_line.forecasted_unit)

        # Calculate standard deviation
        std_deviation = np.std(x)

        if rec.create_date.strftime('%Y-%m') in [line.name for line in sale_forecast_lines]:
          filtered_values = [
            line.fitted_values.values[0] for line in sale_forecast_lines
          ]

          forecast_num = sum(filtered_values)

          # Calculate safety stock
          service_level = 95
          lead_time = config_settings.po_lead
          z_score = norm.ppf(service_level / 100)
```

```
            ss = z_score * math.sqrt(lead_time * std_deviation ** 2)
            ss = round(ss, 0)

            # Calculate max quantity
            reorder_point = (forecast_num * lead_time) + ss
            max_qty = reorder_point + forecast_num - (forecast_num * lead_time)
            max_qty = round(max_qty, 0)

            # Update records
            orderpoint = self.env['stock.warehouse.orderpoint'].search([
               ('product_id', '=', rec.product_id.id)
            ])
            orderpoint.write({
               'qty_forecast': forecast_num,
               'product_min_qty': ss,
               'product_max_qty': max_qty,
            })
```

4.3.2 Penentuan Maximum Quantity

Pada segmen program 4.4, kolom *to order* akan di update jika terdapat perubahan dalam kolom *qty_multiple*, *qty_forecast*, *product_min_qty*, *product_max_qty*, dan atau *qty_on_hand*. Beberapa ketentuan diaplikasikan seperti jika tidak terdapat produk tersebut, maka kolom *to order* ditiadakan. Jika *quantity on hand* lebih dari *quantity forecast* maka kolom *to order* menampilkan angka 0 atau ekuivalen dengan tidak perlu dilakukan order sama sekali. Atau jika *quantity on hand* lebih kecil dari *quantity forecast* dan lebih besar dari *minimum quantity* maka *to order* akan menerima hasil dari *quantity forecast* dikurangi *quantity on hand*. Atau jika *quantity on hand* lebih besar dari 0 dan lebih kecil dari *minimum quantity* maka *to order* akan berisikan hasil dari *quantity forecast* dikurangi *quantity on hand*.

Segmen Program 4.4 Penentuan *Quantity to Order*

```
@api.depends('qty_multiple', 'qty_forecast', 'product_min_qty', 'product_max_qty',
'qty_on_hand')
def _compute_qty_to_order(self):
   for orderpoint in self:
      if not orderpoint.product_id or not orderpoint.location_id:
         orderpoint.qty_to_order = False
         continue
      qty_to_order = 0.0
      rounding = orderpoint.product_uom.rounding
      if orderpoint.qty_on_hand >= orderpoint.qty_forecast:
         qty_to_order = 0
```

```
    elif orderpoint.product_min_qty < orderpoint.qty_on_hand < orderpoint.qty_forecast:
        qty_to_order = orderpoint.qty_forecast - orderpoint.qty_on_hand
    elif 0 < orderpoint.qty_on_hand < orderpoint.product_min_qty:
        qty_to_order = orderpoint.qty_forecast - orderpoint.qty_on_hand
    else:
        if float_compare(orderpoint.qty_forecast, orderpoint.product_min_qty,
precision_rounding=rounding) > 0:
            qty_to_order = max(orderpoint.product_min_qty, orderpoint.product_max_qty) -
orderpoint.qty_forecast

            remainder = orderpoint.qty_multiple > 0 and qty_to_order % orderpoint.qty_multiple or
0.0
            if float_compare(remainder, 0.0, precision_rounding=rounding) > 0:
                qty_to_order += orderpoint.qty_multiple - remainder
    orderpoint.qty_to_order = qty_to_order
```