LAMPIRAN

Lampiran 1 : Kode keseluruhan

- Source Code operational_models.py

```
import datetime
from app import db, bcrypt

class Customer(db.Model):
    __tablename__ = 'customers'
    id = db.Column('CustomerID', db.Integer,
primary_key=True)
    name = db.Column('Name', db.String(255),
nullable=False)
    phone = db.Column('Phone', db.String(15))
    email = db.Column('Email', db.String(255),
unique=True)
    address = db.Column('Address', db.Text)

    # Relationships
    sales = db.relationship('Sale',
back_populates='customer')

    def to_dict(self):
        return {
            'id': self.id,
            'name': self.name,
            'phone': self.phone,
            'email': self.email,
            'address': self.address
        }

class Supplier(db.Model):
    __tablename__ = 'suppliers'
    id = db.Column('SupplierID', db.Integer,
primary_key=True)
    name = db.Column('Name', db.String(255),
nullable=False)
    phone = db.Column('Phone', db.String(15))
    email = db.Column('Email', db.String(255),
unique=True)
    address = db.Column('Address', db.Text)

    # Relationships
    purchases = db.relationship('Purchase',
back_populates='supplier')

    def to_dict(self):
        return {
            'id': self.id,
            'name': self.name,
            'phone': self.phone,
            'email': self.email,
            'address': self.address
        }
```

```
class Category(db.Model):
    __tablename__ = 'categories'
    id = db.Column('CategoryID', db.Integer,
primary_key=True)
    name = db.Column('Name', db.String(255),
nullable=False)
    description = db.Column('Description', db.Text)

    # Relationships
    inventory_items = db.relationship('Inventory',
back_populates='category')

    def to_dict(self):
        return {
            'id': self.id,
            'name': self.name,
            'description': self.description
        }

class Inventory(db.Model):
    __tablename__ = 'inventory'
    id = db.Column('ItemID', db.Integer,
primary_key=True)
    sku = db.Column('SKU', db.String(64),
nullable=False, default='')
    name = db.Column('Name', db.String(255),
nullable=False)
    unit = db.Column('Unit', db.String(64),
nullable=False)
    description = db.Column('Description', db.Text)
    category_id = db.Column('CategoryID',
db.Integer, db.ForeignKey('categories.CategoryID'))
    current_stock = db.Column('CurrentStock',
db.Integer, nullable=False, default=0)
    reorder_level = db.Column('ReorderLevel',
db.Integer)
    unit_price = db.Column('UnitPrice',
db.Numeric(10, 2), nullable=False, default=0.0)

    # Relationships
    category = db.relationship('Category',
back_populates='inventory_items')
    purchase_details =
db.relationship('PurchaseDetail',
back_populates='item')
    sale_details = db.relationship('SaleDetail',
back_populates='item')

    def to_dict(self):
        return {
            'id': self.id,
            'sku': self.sku,
            'name': self.name,
            'unit': self.unit,
```

```
            'description': self.description,
            'category_id': self.category_id,
            'current_stock': self.current_stock,
            'reorder_level': self.reorder_level,
            'unit_price': self.unit_price
        }

class Purchase(db.Model):
    __tablename__ = 'purchases'
    id = db.Column('PurchaseID', db.Integer,
primary_key=True)
    supplier_id = db.Column('SupplierID',
db.Integer, db.ForeignKey('suppliers.SupplierID'))
    purchase_date = db.Column('PurchaseDate',
db.DateTime, default=datetime.datetime.utcnow)
    total_amount = db.Column('TotalAmount',
db.Numeric(10, 2))

    # Relationships
    supplier = db.relationship('Supplier',
back_populates='purchases')
    purchase_details =
db.relationship('PurchaseDetail',
back_populates='purchase')

    def to_dict(self):
        return {
            'id': self.id,
            'supplier_id': self.supplier_id,
            'purchase_date': self.purchase_date,
            'total_amount': self.total_amount,
            'supplier': self.supplier.to_dict(),
            'details': [detail.to_dict() for detail
in self.purchase_details]
        }

class PurchaseDetail(db.Model):
    __tablename__ = 'purchase_details'
    id = db.Column('PurchaseDetailID', db.Integer,
primary_key=True)
    purchase_id = db.Column('PurchaseID',
db.Integer, db.ForeignKey('purchases.PurchaseID'))
    item_id = db.Column('ItemID', db.Integer,
db.ForeignKey('inventory.ItemID'))
    quantity = db.Column('Quantity', db.Integer,
nullable=False)
    unit_price = db.Column('UnitPrice',
db.Numeric(10, 2), nullable=False)

    # Relationships
    purchase = db.relationship('Purchase',
back_populates='purchase_details')
    item = db.relationship('Inventory',
back_populates='purchase_details')
```

```
    def to_dict(self):
        return {
            'id': self.id,
            'purchase_id': self.purchase_id,
            'item_id': self.item_id,
            'quantity': self.quantity,
            'unit_price': self.unit_price,
            'item': self.item.to_dict()
        }

class Sale(db.Model):
    __tablename__ = 'sales'
    id = db.Column('SaleID', db.Integer,
primary_key=True)
    customer_id = db.Column('CustomerID',
db.Integer, db.ForeignKey('customers.CustomerID'))
    sale_date = db.Column('SaleDate', db.DateTime,
default=datetime.datetime.utcnow)
    total_amount = db.Column('TotalAmount',
db.Numeric(10, 2))

    # Relationships
    customer = db.relationship('Customer',
back_populates='sales')
    sale_details = db.relationship('SaleDetail',
back_populates='sale')

    def to_dict(self):
        return {
            'id': self.id,
            'customer_id': self.customer_id,
            'sale_date': self.sale_date,
            'total_amount': self.total_amount
        }

class SaleDetail(db.Model):
    __tablename__ = 'sale_details'
    id = db.Column('SaleDetailID', db.Integer,
primary_key=True)
    sale_id = db.Column('SaleID', db.Integer,
db.ForeignKey('sales.SaleID'))
    item_id = db.Column('ItemID', db.Integer,
db.ForeignKey('inventory.ItemID'))
    quantity = db.Column('Quantity', db.Integer,
nullable=False)
    unit_price = db.Column('UnitPrice',
db.Numeric(10, 2), nullable=False)

    # Relationships
    sale = db.relationship('Sale',
back_populates='sale_details')
    item = db.relationship('Inventory',
back_populates='sale_details')

    def to_dict(self):
```

```
        return {
            'id': self.id,
            'sale_id': self.sale_id,
            'item_id': self.item_id,
            'quantity': self.quantity,
            'unit_price': self.unit_price
        }

class Forecast(db.Model):
    __tablename__ = 'forecasts'
    id = db.Column(db.Integer, primary_key=True)
    date = db.Column(db.Date, nullable=False,
unique=True)
    purchase_forecast = db.Column(db.Float,
nullable=False)
    sale_forecast = db.Column(db.Float,
nullable=False)
```

- Source Code erp_models.py

```
import datetime
from app import db, bcrypt

class RolePrivilege(db.Model):
    """
    Association table for many-to-many relationship
between Roles and Privileges.
    """
    __tablename__ = 'role_privileges'
    role_id = db.Column(db.Integer,
db.ForeignKey('roles.id'), primary_key=True)
    privilege_id = db.Column(db.Integer,
db.ForeignKey('privileges.id'), primary_key=True)

    def to_dict(self):
        return {
            'role_id': self.role_id,
            'privilege_id': self.privilege_id
        }

class Role(db.Model):
    """
    Represents a user role with a set of privileges.
    """
    __tablename__ = 'roles'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True,
nullable=False)
    description = db.Column(db.String(256),
nullable=True)

    privileges = db.relationship(
        'Privilege',
        secondary='role_privileges',
        backref=db.backref('roles', lazy='dynamic')
```

```
    )

    def to_dict(self):
        return {
            'id': self.id,
            'name': self.name,
            'description': self.description,
            'privileges': [privilege.name for
privilege in self.privileges]
        }

    def __repr__(self):
        return f"<Role {self.name}>"

class Privilege(db.Model):
    """
    Represents a specific privilege (e.g.,
'view_reports', 'manage_users').
    """
    __tablename__ = 'privileges'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True,
nullable=False)
    description = db.Column(db.String(256),
nullable=True)

    def to_dict(self):
        return {
            'id': self.id,
            'name': self.name,
            'description': self.description
        }

    def __repr__(self):
        return f"<Privilege {self.name}>"

class User(db.Model):
    """
    Represents a user in the system.
    """
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), unique=True,
nullable=False)
    email = db.Column(db.String(120), unique=True,
nullable=False)
    password_hash = db.Column(db.String(128),
nullable=False)
    role_id = db.Column(db.Integer,
db.ForeignKey('roles.id'), nullable=False)

    role = db.relationship('Role',
backref=db.backref('users', lazy='dynamic'))

    def set_password(self, password):
```

```
        """Hashes and stores the password."""
        self.password_hash =
bcrypt.generate_password_hash(password).decode('utf-
8')

    def check_password(self, password):
        """Checks the hashed password."""
        return
bcrypt.check_password_hash(self.password_hash,
password)

    def has_privilege(self, privilege_name):
        """Checks if the user has a specific
privilege."""
        return any(privilege.name == privilege_name
for privilege in self.role.privileges)

    def to_dict(self):
        return {
            'id': self.id,
            'username': self.username,
            'email': self.email,
            'role': self.role.name
        }

    def __repr__(self):
        return f"<User {self.username}>"

class Session(db.Model):
    """
    Represents a user session.
    """
    __tablename__ = 'sessions'
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer,
db.ForeignKey('users.id'), nullable=False)
    session_key = db.Column(db.String(128),
unique=True, nullable=False)
    created_at = db.Column(db.DateTime,
default=datetime.datetime.utcnow, nullable=False)
    expires_at = db.Column(db.DateTime,
nullable=False)

    user = db.relationship('User',
backref=db.backref('sessions', lazy='dynamic'))

    def is_active(self):
        """Check if the session is still valid."""
        return datetime.datetime.utcnow() <
self.expires_at

    def to_dict(self):
        return {
            'id': self.id,
            'user_id': self.user_id,
```

```
            'session_key': self.session_key,
            'created_at': self.created_at,
            'expires_at': self.expires_at
        }

    def __repr__(self):
        return f"<Session user_id={self.user_id}
expires_at={self.expires_at}>"
```

- Source Cod init.py

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
from flask_session import Session
from flask_bcrypt import Bcrypt
from flask_cors import CORS, cross_origin
from apscheduler.schedulers.background import
BackgroundScheduler
from app.config import Config

# Initialize extensions
db = SQLAlchemy()
migrate = Migrate()
bcrypt = Bcrypt()
session_manager = Session()
cors = CORS()

def create_app(config_class=Config):
    """
    Application Factory for creating and configuring
the Flask app.
    """
    app = Flask(__name__)
    app.config.from_object(config_class)

    # Initialize extensions
    cors.init_app(app, supports_credentials=True,
resources={r"*": {"origins": "*"}})
    db.init_app(app)
    migrate.init_app(app, db)
    bcrypt.init_app(app)
    session_manager.init_app(app)

    # Register blueprints
    from app.routes.auth import auth_bp
    from app.routes.main import main_bp
    from app.routes.admin import admin_bp
    from app.routes.acc_control import acc_control
    from app.routes.operational.sales import sale_bp
    from app.routes.operational.purchase import
purchase_bp
    from app.routes.operational.inventory import
inventory_bp
```

```
    from app.routes.operational.supplier import
supplier_bp
    from app.routes.operational.customer import
customer_bp
    from app.routes.operational.forecast import
forecast_bp

    from app.routes.operational.forecast import
perform_forecast


    app.register_blueprint(auth_bp,
url_prefix='/auth')
    app.register_blueprint(main_bp,
url_prefix='/main')
    app.register_blueprint(admin_bp,
url_prefix='/admin')
    app.register_blueprint(acc_control,
url_prefix='/acc_control')
    app.register_blueprint(sale_bp,
url_prefix='/sales')
    app.register_blueprint(purchase_bp,
url_prefix='/purchases')
    app.register_blueprint(inventory_bp,
url_prefix='/inventory')
    app.register_blueprint(supplier_bp,
url_prefix='/suppliers')
    app.register_blueprint(customer_bp,
url_prefix='/customers')
    app.register_blueprint(forecast_bp,
url_prefix='/forecast')

    @app.route('/')
    @cross_origin(supports_credentials=True)
    def welcome_page():
        return render_template('welcome.html')

    # Register custom error handlers
    register_error_handlers(app)

    # Initialize and start the scheduler
    scheduler = BackgroundScheduler()
    scheduler.add_job(func=perform_forecast,
trigger="interval", days=1)
    scheduler.start()

    return app

def register_error_handlers(app):
    """
    Register custom error handlers for the
application.
    """
    @app.errorhandler(404)
    def not_found_error(error):
```

```
        return {"error": "Page not found"}, 404

    @app.errorhandler(500)
    def internal_error(error):
        db.session.rollback()
        return {"error": "Internal server error"},
500
```

- Source Code fifo_utils.py

```
from app.models.operational_models import Purchase,
SaleDetail, PurchaseDetail
from sqlalchemy import and_, desc
from datetime import datetime

def calculate_profit(sale_id):
    """
    Calculates profit by using the last purchase
price as the cost basis
    """
    sale_details =
SaleDetail.query.filter_by(sale_id=sale_id).all()
    total_revenue = 0
    total_cost = 0

    for detail in sale_details:
        # Calculate revenue
        revenue = detail.quantity *
float(detail.unit_price)
        total_revenue += revenue

        # Get the most recent purchase price for
this item
        last_purchase =
PurchaseDetail.query.join(Purchase)\
            .filter(PurchaseDetail.item_id ==
detail.item_id)\
            .order_by(desc(Purchase.purchase_date))\
            .first()

        if last_purchase:
            cost = detail.quantity *
float(last_purchase.unit_price)
        else:
            # Fallback if no purchase history
            cost = 0

        total_cost += cost

    return total_revenue - total_cost


def get_item_cogs(item_id, quantity,
before_date=None):
```

```
    """
    Helper function to calculate COGS for a specific
item quantity using FIFO
    """
    if before_date is None:
        before_date = datetime.now()

    # Get relevant purchases
    purchases =
PurchaseDetail.query.join(PurchaseDetail.purchase).f
ilter(
        and_(
            PurchaseDetail.item_id == item_id,
            Purchase.purchase_date <= before_date
        )
    ).order_by(Purchase.purchase_date.asc()).all()

    cogs = 0
    remaining_quantity = quantity

    for purchase in purchases:
        if remaining_quantity <= 0:
            break

        quantity_from_purchase =
min(remaining_quantity, purchase.quantity)
        purchase_cost = quantity_from_purchase *
float(purchase.unit_price)  # Calculate total cost
        cogs += purchase_cost
        remaining_quantity -= quantity_from_purchase

    # Handle any remaining quantity using the latest
purchase price
    if remaining_quantity > 0 and purchases:
        latest_purchase = purchases[-1]
        cogs += remaining_quantity *
float(latest_purchase.unit_price)

    return cogs
```

- Source Code forecast.py

```
from flask import Blueprint, jsonify, request
from app.models.operational_models import Purchase,
PurchaseDetail, Sale, SaleDetail, Forecast,
Inventory
from sklearn.linear_model import LinearRegression
from sklearn.cluster import KMeans
from app import db
import numpy as np
import datetime
from sqlalchemy.orm import joinedload

forecast_bp = Blueprint('forecast', __name__)
```

```
@forecast_bp.route('/forecast', methods=['GET'])
def get_forecast():
    forecast =
Forecast.query.order_by(Forecast.date.desc()).first(
)
    if forecast:
        return jsonify({
            'purchase_forecast':
forecast.purchase_forecast,
            'sale_forecast': forecast.sale_forecast
        })
    else:
        forecast = perform_forecast(days=7)
        return jsonify({
            'purchase_forecast':
forecast.purchase_forecast,
            'sale_forecast': forecast.sale_forecast
        })

def prepare_data(data, past_date):
    if not data:
        return np.array([]).reshape(-1, 1),
np.array([])
    dates = [(d[0] - past_date).days for d in data]
    quantities = [d[1] for d in data]
    return np.array(dates).reshape(-1, 1),
np.array(quantities)

def perform_forecast(days=7):
    today = datetime.date.today()
    past_date = today -
datetime.timedelta(days=days)
    past_datetime =
datetime.datetime.combine(past_date,
datetime.time.min)

    purchase_details = (
        db.session.query(PurchaseDetail, Purchase)
        .join(Purchase, PurchaseDetail.purchase_id
== Purchase.id)
        .filter(Purchase.purchase_date >=
past_datetime)
        .all()
    )
    sale_details = (
        db.session.query(SaleDetail, Sale)
        .join(Sale, SaleDetail.sale_id == Sale.id)
        .filter(Sale.sale_date >= past_datetime)
        .all()
    )

    purchase_data = [(purchase.purchase_date,
detail.quantity) for detail, purchase in
purchase_details]
```

```
    sale_data = [(sale.sale_date, detail.quantity)
for detail, sale in sale_details]

    purchase_dates, purchase_quantities =
prepare_data(purchase_data, past_datetime)
    sale_dates, sale_quantities =
prepare_data(sale_data, past_datetime)

    purchase_forecast, sale_forecast = 0, 0
    model = LinearRegression()

    if len(purchase_dates) > 0 and
len(purchase_quantities) > 0:
        model.fit(purchase_dates,
purchase_quantities)
        purchase_forecast =
float(model.predict(np.array([[days]]))[0])

    if len(sale_dates) > 0 and len(sale_quantities)
> 0:
        model.fit(sale_dates, sale_quantities)
        sale_forecast =
float(model.predict(np.array([[days]]))[0])

    forecast = Forecast(
        date=today,
        purchase_forecast=purchase_forecast,
        sale_forecast=sale_forecast
    )

    try:
        db.session.add(forecast)
        db.session.commit()
    except Exception as e:
        db.session.rollback()
        raise e

    return forecast

@forecast_bp.route('/items', methods=['GET'])
def get_item_forecast():
    items = Inventory.query.all()
    forecasts = {}
    for item in items:
        forecasts[item.name] =
perform_item_forecast(item.id, days=7)
    return jsonify(forecasts)

def perform_item_forecast(item_id, days=7):
    today = datetime.date.today()
    past_date = today -
datetime.timedelta(days=days)
    past_datetime =
datetime.datetime.combine(past_date,
datetime.time.min)
```

```
    purchase_details = (
        db.session.query(PurchaseDetail, Purchase)
        .join(Purchase, PurchaseDetail.purchase_id
== Purchase.id)
        .filter(Purchase.purchase_date >=
past_datetime, PurchaseDetail.item_id == item_id)
        .all()
    )
    sale_details = (
        db.session.query(SaleDetail, Sale)
        .join(Sale, SaleDetail.sale_id == Sale.id)
        .filter(Sale.sale_date >= past_datetime,
SaleDetail.item_id == item_id)
        .all()
    )

    purchase_data = [(purchase.purchase_date,
detail.quantity) for detail, purchase in
purchase_details]
    sale_data = [(sale.sale_date, detail.quantity)
for detail, sale in sale_details]

    purchase_dates, purchase_quantities =
prepare_data(purchase_data, past_datetime)
    sale_dates, sale_quantities =
prepare_data(sale_data, past_datetime)

    purchase_forecast, sale_forecast = 0, 0
    model = LinearRegression()

    if len(purchase_dates) > 0 and
len(purchase_quantities) > 0:
        model.fit(purchase_dates,
purchase_quantities)
        purchase_forecast =
float(model.predict(np.array([[days]]))[0])

    if len(sale_dates) > 0 and len(sale_quantities)
> 0:
        model.fit(sale_dates, sale_quantities)
        sale_forecast =
float(model.predict(np.array([[days]]))[0])

    return {
        'purchase_forecast': purchase_forecast,
        'sale_forecast': sale_forecast
    }

@forecast_bp.route('/clusters', methods=['GET'])
def get_item_clusters():
    sale_details = SaleDetail.query.all()
    item_quantities = {}

    for detail in sale_details:
```

```
        if detail.sale_id not in item_quantities:
            item_quantities[detail.sale_id] = {}
        if detail.item_id not in
item_quantities[detail.sale_id]:

item_quantities[detail.sale_id][detail.item_id] = 0

item_quantities[detail.sale_id][detail.item_id] +=
detail.quantity

    transactions = list(item_quantities.values())
    item_ids = list(set(item_id for transaction in
transactions for item_id in transaction.keys()))
    item_index = {item_id: idx for idx, item_id in
enumerate(item_ids)}

    data = np.zeros((len(transactions),
len(item_ids)))
    for i, transaction in enumerate(transactions):
        for item_id, quantity in
transaction.items():
            data[i, item_index[item_id]] = quantity

    kmeans = KMeans(n_clusters=5)
    kmeans.fit(data)
    clusters = kmeans.labels_

    item_names = {item.id: item.name for item in
Inventory.query.filter(Inventory.id.in_(item_ids)).a
ll()}
    cluster_data = [[] for _ in range(5)]
    for idx, item_id in enumerate(item_ids):
        cluster = clusters[idx]
        item_name = item_names.get(item_id, f"Item
{item_id}")
        cluster_data[cluster].append(item_name)

    return jsonify({
        'clusters': cluster_data
    })
```

Lampiran 2 : Dialog Percakapan

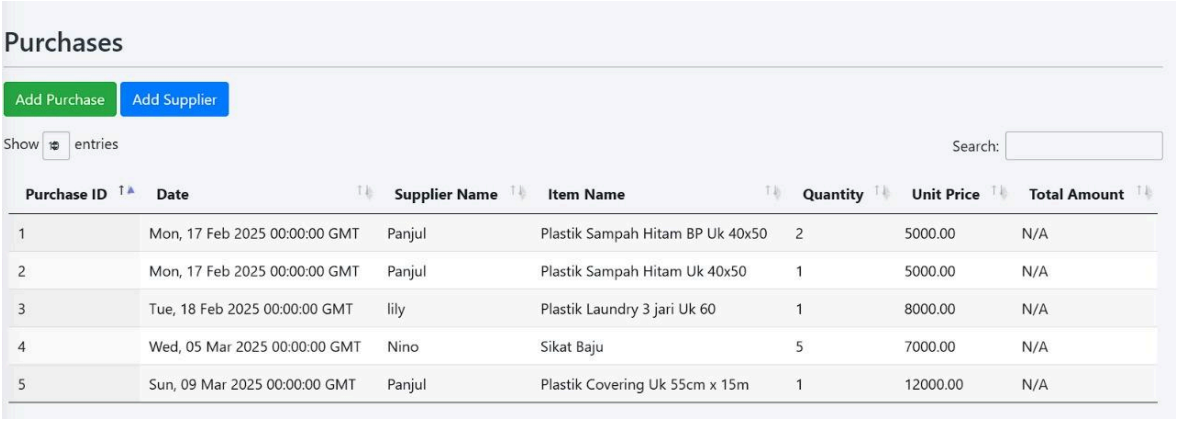
Pembicara	**Percakapan**
Alissa	Selamat sore Ko, perkenalkan nama saya Alissa Salim dari Institut Bisnis dan Informatika Kwik Kian Gie. Saya ingin melakukan wawancara dengan Koko, apakah Koko bisa?
Sefo	Bisa.

Alissa	Pertanyaan 1: Apa saja kegiatan operasional utama yang dilakukan di Toko Gemilang 888?
Sefo	Kegiatan utama di sini adalah menjual plastik, jadi operasionalnya seperti toko pada umumnya.
Alissa	Pertanyaan 2: Sistem atau metode apa yang saat ini digunakan untuk mengelola kegiatan operasional (misalnya stok barang, penjualan, pembelian)?
Sefo	Sistem yang digunakan masih manual, seperti pencatatan barang masuk, pembuatan nota dan surat jalan, serta pencatatan stok barang semuanya dilakukan secara manual. Hal ini menyulitkan dalam mengelola stok barang.
Alissa	Pertanyaan 3: Apa kendala utama yang sering Koko alami dalam pengelolaan operasional toko?
Sefo	Kendala utama adalah dalam mengelola stok barang. Setiap barang yang masuk dan terjual harus dicatat manual. Jika ada pelanggan yang bertanya tentang ketersediaan barang, kami harus mengeceknya langsung. Selain itu, untuk mengetahui total penjualan harian, kami harus merekap secara manual setiap hari.
Alissa	Pertanyaan 4: Apakah toko sudah pernah menggunakan aplikasi atau sistem teknologi sebelumnya? Jika ya, bagaimana pengalaman Koko menggunakannya?
Sefo	Belum pernah menggunakan aplikasi atau sistem teknologi sebelumnya, sehingga saat ini kami ingin membuat sistem untuk mempermudah pengelolaan toko.
Alissa	Pertanyaan 5: Fitur apa saja yang Koko butuhkan dalam sistem ERP untuk mendukung pekerjaan sehari-hari?
Sefo	Fitur yang paling dibutuhkan adalah manajemen stok barang secara real-time, sehingga barang masuk dan keluar bisa terlacak. Dengan sistem ini, kami bisa melihat stok barang yang tersedia dan riwayat barang masuk serta keluar. Selain itu, fitur pencetakan nota penjualan dan surat jalan, serta laporan penjualan harian juga sangat dibutuhkan.
Alissa	Pertanyaan 6: Dalam proses pengelolaan stok barang, apa yang sering menjadi tantangan atau kesulitan?

Sefo	Tantangan utama adalah tidak mengetahui jumlah stok barang secara langsung karena masih menggunakan sistem manual. Pemantauan barang masuk dan keluar juga masih manual, sehingga menyulitkan operasional toko. Selain itu, pencatatan surat jalan dan nota penjualan masih dilakukan secara manual, sehingga laporan harian harus direkap setiap hari.
Alissa	Pertanyaan 7: Seberapa penting bagi Koko untuk mendapatkan laporan atau data operasional secara real-time?
Sefo	Sangat penting. Dengan sistem terkomputerisasi, kami bisa langsung memberitahukan pelanggan tentang ketersediaan barang. Selain itu, laporan penjualan harian dan bulanan bisa dianalisis untuk mengetahui tren penjualan, yang sebelumnya tidak dapat dilakukan karena tidak ada data historis yang terdokumentasi dengan baik.
Alissa	Pertanyaan 8: Apakah Koko merasa sistem baru yang berbasis ERP akan membantu menyelesaikan masalah yang ada saat ini? Jika ya, di bagian mana?
Sefo	Ya, sistem ERP sangat membantu. Dengan sistem ini, stok barang bisa dipantau secara real-time, nota penjualan dapat dicetak secara elektronik, dan penjualan harian bisa langsung diketahui tanpa perlu rekap manual.
Alissa	Pertanyaan 9: Apa yang menjadi prioritas utama Koko dalam sebuah sistem ERP? (Misalnya: kemudahan penggunaan, kecepatan, keamanan, atau lainnya)
Sefo	Prioritas utama adalah kemudahan penggunaan agar karyawan tidak kesulitan dalam mengoperasikan sistem. Selain itu, keamanan juga penting agar data penting toko tidak bocor.
Alissa	Pertanyaan 10: Apakah Koko lebih nyaman dengan sistem berbasis desktop, aplikasi seluler, atau berbasis web? Mengapa?
Sefo	Lebih nyaman menggunakan sistem berbasis web karena bisa diakses dari berbagai komputer yang terhubung dalam jaringan LAN. Jika menggunakan aplikasi seluler, akses menjadi terbatas hanya untuk beberapa orang saja. Dengan sistem berbasis web, komputer mana saja di toko bisa mengakses sistem dengan mudah.

Alissa	Makasih Ko atas wawancara yang sudah disampaikan, sekian dan terima kasih.

Lampiran 3 : Data Forecasting



Purchases

Add Purchase | Add Supplier

Show 10 entries | Search:

Purchase ID	Date	Supplier Name	Item Name	Quantity	Unit Price	Total Amount
1	Mon, 17 Feb 2025 00:00:00 GMT	Panjul	Plastik Sampah Hitam BP Uk 40x50	2	5000.00	N/A
2	Mon, 17 Feb 2025 00:00:00 GMT	Panjul	Plastik Sampah Hitam Uk 40x50	1	5000.00	N/A
3	Tue, 18 Feb 2025 00:00:00 GMT	lily	Plastik Laundry 3 jari Uk 60	1	8000.00	N/A
4	Wed, 05 Mar 2025 00:00:00 GMT	Nino	Sikat Baju	5	7000.00	N/A
5	Sun, 09 Mar 2025 00:00:00 GMT	Panjul	Plastik Covering Uk 55cm x 15m	1	12000.00	N/A

draft_skripsi-41210018-Yehezkiel_Arly.pdf

ORIGINALITY REPORT

15%	14%	3%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	eprints.kwikkiangie.ac.id Internet Source	1%
2	library.binus.ac.id Internet Source	1%
3	docplayer.info Internet Source	1%
4	Submitted to Universitas Brawijaya Student Paper	1%
5	repository.ub.ac.id Internet Source	1%
6	www.coursehero.com Internet Source	1%
7	id.scribd.com Internet Source	<1%
8	repository.dinamika.ac.id Internet Source	<1%
9	text-id.123dok.com Internet Source	<1%

KWIK KIAN GIE
SCHOOL OF BUSINESS

INSTITUT BISNIS DAN INFORMATIKA KWIK KIAN GIE

Jl. Yos Sudarso Kav. 87, Sunter, Jakarta Utara 14350

KARTU PROSES BIMBINGAN KARYA AKHIR

Nomor Dokumen	Tanggal Pembuatan	Tanggal Revisi	Disahkan Oleh
IBIKKG/SPMI/FORM-A.04.06.02-01	Nopember 2017	Edisi 1: Nopember 2017	AREA/UPT/LPM

Nama : Yobeakel Ady NIM : 41210018

Konsentrasi : Business Intelligence Topik : Aplikasi Berbasis Web

KRS/Registrasi : Semester Ganjil/~~Genap~~*) Th. Akademik : 2024 / 2025

Dosen Pembimbing : Joko Susilo, S.Kom., M.M., M.Kom.

Judul : Implementasi Sistem Enterprise Resource Planning Pada Toko Gemilang 888

No	Tanggal	Pokok Bahasan	Paraf Mahasiswa	Paraf Dosen	Catatan
1.	20/11/2024	Bab I			Kwik
2.	3/12/2024	Bab II			Revisi, lanjut ke
3.	11/12/2024	Bab III			Revisi
4.	20/12/2024	Bab I-III			ok
5	8/1/2025	Bab I, II, III			ok.
6	22/1/2025	BAB IV			Revisi → DPD
7	10/02/2025	BAB IV, V			Revisi
8	17/02/2025	BAB IV, V			ok

Catatan: - *) Coret yang tidak perlu

- Bimbingan minimal 8 kali pertemuan dan mencantumkan tanda tangan pada setiap kali bimbingan sebagai bukti telah melakukan proses bimbingan

GEMILANG 888

MENYEDIAKAN KANTONG PLASTIK, HD PUTIH, HD MERAH, PLASTIK SAMPAH, PLASTIK PACKING, DLL

Kontak : 087884392691
Alamat : Cempaka. Sari 3 No.12. RT.12/RW.8. Harapan Mulya. Kec. Kemayoran, Kota Jakarta Pusat.

SURAT KETERANGAN PENELITIAN

Yang bertanda tangan di bawah ini Kong Sie Fo, selaku owner dari Toko Gemilang 888
Dengan ini menerangkan bahwa :

Nama	: Yehezkiel Arly
NIM	: 41210018
Program Studi	: Sistem Informasi
Perguruan Tinggi	: Institut Bisnis dan Informatika Kwik Kian Gie

Telah melakukan penelitian di Toko Plastik Gemilang 888 dalam rangka penyusunan skripsi dengan judul: "Implementasi Sistem Enterprise Resource Planning Pada Toko Gemilang 888". Penelitian ini dilakukan dalam kurun waktu 21 Oktober 2024 hingga 12 Febuari 2025.

Demikian surat keterangan ini dibuat untuk dipergunakan sebagaimana mestinya.

Jakarta, 28 Febuari 2025
Toko Plastik Gemilang 888

Kong Sie Fo
Owner Toko Gemilang 888

KWIK KIAN GIE
SCHOOL OF BUSINESS
d/h Institut Bisnis dan Informatika Indonesia (IBII)

SURAT PERNYATAAN

Saya yang bertandatangan di bawah ini :

Nama : Yehezkiel Arly

Program Studi : Sistem Informasi

NIM : 41210018

Alamat Lengkap : Jl. Industri Dalam Gang Q no.21 RT 10 / RW 14 (Tinggal) 14420

TMN Surya 2 E-5/30 (KTP)

Kode pos : 11830

Telp Kantor : (021) 2957 8888

Telp Rumah : -

No. HP : +62 821 1277 4927

Menyatakan dengan sungguh-sungguh bahwa :

1. Keabsahan data dan hal-hal lain yang berkenaan dengan keaslian dalam penyusunan karya akhir ini merupakan tanggung jawab pribadi.

2. Apabila dikemudian hari timbul masalah dengan keabsahan data dan keaslian/originalitas karya akhir adalah di luar tanggung jawab Institut Bisnis Dan Informatika Kwik Kian Gie dan saya bersedia menanggung segala resiko sanksi yang dikeluarkan Institut Bisnis Dan Informatika Kwik Kian Gie dan gugatan yang oleh pihak lain yang merasa dirugikan.

Demikian agar yang berkepentingan maklum

Jakarta, 22 Maret 2025

Yang membuat pernyataan

Yehezkiel Arly

(Nama Lengkap)

KWIK KIAN GIE
SCHOOL OF BUSINESS

PROGRAM SARJANA

(Beri tanda √ pada program studi yang Anda pilih)

- ☐ Program Studi Manajemen
- ☐ Program Studi Akuntansi
- ☐ Program Studi Administrasi Bisnis
- ☐ Program Studi Ilmu Komunikasi
- ☑ Program Studi Sistem Informasi
- ☐ Program Studi Teknik Informatika

FORMULIR PERSETUJUAN PERBAIKAN SKRIPSI / KARYA AKHIR

Nama Mahasiswa : Yehezkiel Arly NIM : 41210018
Pembimbing : Joko Susila, S.Kom, M.M., M.Kom Tanggal Sidang : 18/03/2025
Judul Skripsi : Implementasi Sistem Enterprise Resource Planning Pada Toko Gemilang 888

No.	Hal yang harus diperbaiki	STATUS PERBAIKAN (Check List =V)	Persetujuan Salah Satu Penguji (Boleh dipilih salah satu) Tanda Tangan dan Nama		Tanggal Persetujuan
			Penguji I	Penguji II	
1.	Abstrak	✓			21/03/2025
2.	Daftar Pustaka	✓			~11~
3.	Perbesar gambar diagram	✓			~4~
4.	Menghilangkan teori Machine Learning	✓			~11~
5.	Istilah asing dgn tulisan miring	✓			~11~
6.	Tambah Hasil Forecast	✓			~11~
7.	Judul Tabel Diatas	✓			~11~
8.					
9.					
Dsb					

Catatan:
1 Revisi skripsi / karya akhir dapat dilakukan oleh Penguji I atau Penguji II (salah satu)
2. Status Perbaikan (Check List = V) diisi atau dilengkapi oleh Tim Penguji
3. Pengesahan akhir dari skripsi / karya akhir di sah kan oleh Pembimbing,
4. Formulir ini dibawa mahasiswa saat meminta pengesahaan Dosen Pembimbing.
5. Formulir ini dilampirkan di bagian paling belakang skripsi / Karya Akhir