Appendix 4 (Code of the program)

main.py

```python
from app import create_app

app = create_app()

if __name__ == "__main__":
    # Only for debugging while developing
    app.run(host="0.0.0.0", debug=True, port=8080)
```

config.py

```python
from os import environ, path
from dotenv import load_dotenv

BASE_DIR = path.abspath(path.dirname(__file__))
load_dotenv(path.join(BASE_DIR, ".env"), override=True)


class Config:
    """Flask configuration variables."""

    # General Config
    APP_NAME = environ.get("APP_NAME")
    ENV = environ.get("FLASK_ENV")
    DEBUG = environ.get("FLASK_DEBUG")
    SECRET_KEY = environ.get("SECRET_KEY")

    # Static Assets
    STATIC_FOLDER = "static"
    TEMPLATE_FOLDER = "templates"

    # Database
    SQLALCHEMY_DATABASE_URI = environ.get("SQLALCHEMY_DATABASE_URI")
    SQLALCHEMY_TRACK_MODIFICATIONS = environ.get("SQLALCHEMY_TRACK_MODIFICATIONS")
```

__init__.py

```python
from flask import Flask
from flask_login import login_required
from dash.long_callback import DiskcacheLongCallbackManager
import diskcache


def create_app():
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_object("config.Config")

    register_blueprints(app)
    register_dashapps(app)
    register_extensions(app)

    return app


def register_blueprints(app):
    from .routes import routes
    from .auth import auth

    app.register_blueprint(routes, url_prefix="/")
    app.register_blueprint(auth, url_prefix="/")


def register_dashapps(app):
    from .dash import (
        price_index,
        analogs,
        data_table,
    )

    # Meta tags for viewport responsiveness
    meta_viewport = {
        "name": "viewport",
        "content": "width=device-width, initial-scale=1, shrink-to-fit=no",
    }

    cache = diskcache.Cache("./cache")
    lcm = DiskcacheLongCallbackManager(cache)

    with app.app_context():
        app = analogs.init_dash(app)
        app = data_table.init_dash(app)
        app = price_index.init_dash(
            app,
            meta_viewport,
            lcm,
        )
    _protect_dashviews(app)


```

auth.py

```python
from flask import Blueprint, render_template, request, flash, redirect, url_for
from .models import User
from werkzeug.security import check_password_hash, generate_password_hash
from app.extensions import db
from flask_login import current_user, login_required, login_user, logout_user

auth = Blueprint("auth", __name__)

# login page route
@auth.route("/login", methods=["GET", "POST"])
def login():
    # redirecting to home page if user is already logged in
    if current_user.is_authenticated:
        return redirect(url_for("routes.index"))

    # checking whether a request is post, to prevent unwanted requests
    if request.method == "POST":
        # getting data from the form
        username = request.form.get("username")
        password = request.form.get("password")
        remember = request.form.get("remember")
        if remember == "on":
            remember = True
        else:
            remember = False

        try:
            user = User.query.filter_by(username=username).first()
        except Exception:
            flash(["Invalid username or password"], category="error")
            return redirect(url_for("auth.login"))
        # checking whether user exists in the database
        if not user:
            flash(["Invalid username or password"], category="error")
            return redirect(url_for("auth.login"))
        # checking whether the password matches with the password in the database
        if check_password_hash(user.password_hash, password):
            login_user(user, remember=remember)
            # redirecting to the main page if password matches
            return redirect(url_for("routes.index"))
        else:
            # flashing error message and redirecting if password mismatches
            flash(["Invalid username or password"], category="error")
            return redirect(url_for("auth.login"))

    return render_template("login.html")


```

```python
@auth.route("/logout")
@login_required
def logout():
    logout_user()
    return redirect(url_for("auth.login"))


@auth.route("/register", methods=["GET", "POST"])
def register():
    # checking if user is authenticated
    if current_user.is_authenticated:
        return redirect(url_for("routes.index"))

    if request.method == "POST":
        username = request.form.get("username")
        password1 = request.form.get("password1")
        password2 = request.form.get("password2")

        # checking if the password has proper characters is not too short or too long
        if len(username) < 4:
            flash(["Username must be at least 4 characters long"], category="error")
        elif len(username) ⩾ 15:
            flash(["Username must be at most 15 characters long"], category="error")
        elif password1 ≠ password2:
            flash(["Passwords do not match"], category="error")
        elif len(password1) ⩽ 5:
            flash(["Password must be at least 5 characters long"], category="error")
        elif len(password1) ⩾ 15:
            flash(["Password must be at atmost  15 characters long"], category="error")
        elif not (
            any([x.isupper() for x in password1])
            and any([x.islower() for x in password1])
            and any([x.isdigit() for x in password1])
        ):
            flash(
                [
                    "Password must contain:",
                    "    - at least one capital letter",
                    "    - at least a single number",
                ],
                category="error",
            )
        else:
            # new userr is created and added to database
            new_user = User(
                username=username,
                password_hash=generate_password_hash(password1, method="sha256"),
            )
```

```
            db.session.add(new_user)
            db.session.commit()
            login_user(new_user)
            flash("Registration successful", category="success")
            # redirecting to home page
            return redirect(url_for("routes.index"))

    return render_template("register.html")
```

extensions.py

```
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager
from flask_migrate import Migrate

db = SQLAlchemy()
migrate = Migrate(compare_type=True)
login = LoginManager()
```

models.py

```
from flask_login import UserMixin
from werkzeug.security import check_password_hash
from werkzeug.security import generate_password_hash
from sqlalchemy.sql import func

from app.extensions import db
from app.extensions import login

from sqlalchemy import or_, UniqueConstraint
from sqlalchemy.orm import foreign, remote


@login.user_loader
def load_user(id):
    return User.query.get(int(id))


class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), index=True, unique=True)
    password_hash = db.Column(db.String(128))

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

```

```
    def check_password(self, password):
        return check_password_hash(self.password_hash, password)

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


class Product(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(256), unique=True, nullable=False)
    url = db.Column(db.String(256), unique=True, nullable=False)
    manufacturer_id = db.Column(
        db.Integer, db.ForeignKey("manufacturer.id"), nullable=False
    )
    eshop_id = db.Column(db.Integer, db.ForeignKey("eshop.id"),
nullable=False)
    store = db.relationship("Store", backref="product")

    analogs = db.relationship(
        "Analog",
        primaryjoin=lambda: or_(
            Analog.id == foreign(remote(Analog.product_id_1)),
            Analog.id == foreign(remote(Analog.product_id_2)),
        ),
        viewonly=True,
    )

    def __repr__(self):
        return "<Product {}>".format(self.name)


```

```
class Manufacturer(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True, nullable=False)
    products = db.relationship("Product", backref="manufacturer", lazy=True)

    def __repr__(self):
        return "<Manufacturer {}>".format(self.name)


class Eshop(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True, nullable=False)
    products = db.relationship("Product", backref="eshop", lazy=True)

    def __repr__(self):
        return "<Eshop {}>".format(self.name)
```

```python
class Store(db.Model):
    __tablename__ = "store"
    # __table_args__ = (UniqueConstraint("product_id", "date"),)

    product_id = db.Column(db.Integer, db.ForeignKey("product.id"), nullable=False)
    price = db.Column(
        db.Float, nullable=False
    )  # primary_key just to not raise errors ...
    date = db.Column(db.DateTime(timezone=True), nullable=False, default=func.now())

    __mapper_args__ = {"primary_key": [product_id, date]}


class Analog(db.Model):
    id = db.Column(
        db.Integer,
        primary_key=True,
    )
    product_id_1 = db.Column(db.Integer, db.ForeignKey("product.id"))
    product_id_2 = db.Column(db.Integer, db.ForeignKey("product.id"))

    product_1 = db.relationship("Product", foreign_keys=product_id_1)
    product_2 = db.relationship("Product", foreign_keys=product_id_2)
```

routes.py

```python
from flask import render_template
from flask import Blueprint
from flask_login import current_user, login_required, login_user, logout_user

routes = Blueprint("routes", __name__)


@routes.route("/")
@login_required
def index():
    return render_template(
        "index.html",
        content="You are logged in! You can access the following pages:",
        user=current_user,
    )


@routes.route("/update-data")
@login_required
def update_data():
    pass
```

crawler.py

```python
import httpx
from lxml import html
import pandas as pd
from config import Config
from sqlalchemy import create_engine, text

engine = create_engine(Config.SQLALCHEMY_DATABASE_URI, client_encoding="utf8")

# Abstract Class for Crawlers
class Crawler:
    def __init__(self):
        pass

    def get_link(self, url):
        r = httpx.get(
            url,
            headers={
                "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36"
            },
        )
        r.raise_for_status()  # raise an error if status_code ≠ 200
        content = html.fromstring(r.content)
        return content

    def crawl(self):
        pass

    def save(self, df):
        # start connection with database
        with engine.begin() as conn:

            sql_list = [f"('{eshop}')" for eshop in df.eshop.unique()]
            # generate sql sequences
            sql_str = ",".join(sql_list)
            # insert unique eshops to database
            conn.execute(
                f"""
                INSERT INTO eshop (name)
                VALUES {sql_str}
                ON CONFLICT (name)
                DO NOTHING;
                """
            )
```

```
            sql_list = [
                f"('{manufacturer}')" for manufacturer in df.manufacturer.unique()
            ]
            sql_str = ",".join(sql_list)
            # insert unique manufacturers to database
            conn.execute(
                f"""
                INSERT INTO manufacturer (name)
                VALUES {sql_str}
                ON CONFLICT (name)
                DO NOTHING;
                """
            )

            sql_list = [
                f"""('{title.replace("'", "''")}', '{url.replace("'", "''")}', '{manufacturer}', '{eshop}')"""
                for title, manufacturer, eshop, url, price in list(
                    df.itertuples(index=False, name=None)
                )
            ]
            sql_str = ",".join(sql_list)
            # insert unique products to database
            conn.execute(
                text(
                    f"""
                WITH inputvalues(name, url, manufacturer, eshop) AS (
                    VALUES {sql_str}
                )
                INSERT INTO product (name, url, manufacturer_id, eshop_id)
                SELECT d.name, d.url, manufacturer.id, eshop.id
                FROM inputvalues as d
                INNER JOIN eshop ON eshop.name = d.eshop
                INNER JOIN manufacturer ON manufacturer.name = d.manufacturer
                ON CONFLICT
                DO NOTHING;
                """
                )
            )

            sql_list = [
                f"""((SELECT id FROM product WHERE name='{title.replace("'", "''")}'), {price}, now())"""
                for title, manufacturer, eshop, url, price in list(
                    df.itertuples(index=False, name=None)
                )
            ]
            sql_str = ",".join(sql_list)
```

```
            # insert the data of current prices of products into database
            result = conn.execute(
                text(
                    f"""
                WITH inputvalues(id, price, date) AS (
                    VALUES {sql_str}
                )
                INSERT INTO store (product_id, price, date)
                SELECT d.id, d.price, d.date
                FROM inputvalues as d
                WHERE d.id IS NOT NULL;
                """
                )
            )
            print(f"Result: {result}\nlastrowid: {result.lastrowid}")
```