Det medfølgende projekt fremviser nogle af de ting der er blevet bedt om i opgaven i forhold til designmønstre og koncepter inden for softwareudvikling.

Ud over det er der skabt to UML-diagrammer, et diagram som beskriver klasserne i forhold til hvordan det skal se ud som design og et sekvensdiagram der viser hvordan klasser arbejder sammen når programmet bliver brugt.

Her er en mere detaljeret beskrivelse af hvad koden består af:

1. **Fabriksmetode-mønsteret**: Klassen **ProductFactory** demonstrerer Fabriksmetode-mønsteret. Den indekapsler logikken til at oprette forskellige typer produktobjekter baseret på en angivet type. Metoden **create\_item** indlæser dynamisk klasser fra modulet **Item** og opretter instanser af specifikke produkttyper som bærbare computere, opladere, skærme og stationære computere.
2. **Singleton-mønsteret**: Klassen **Database** udviser Singleton-mønsteret. Den sikrer, at der kun eksisterer én instans af databasen i hele applikationens levetid. Den singleton-instans tilgås gennem en statisk metode, hvilket tillader global adgang til databasen og forhindrer, at der oprettes flere instanser.
3. **Logging**: Koden implementerer logging-funktionalitet ved hjælp af Pythons logging-modul. Dekoratoren **log\_action** logger forskellige databaseoperationer såsom indsættelse, hentning og sletning af data. Denne logningsmekanisme giver synlighed ind i databaseaktiviteterne og hjælper med fejlfinding, revision og overvågning.
4. **Arv og polymorfi**: Modulet **Item** demonstrerer arv og polymorfi. Basisklassen **Item** definerer fælles attributter og metoder, som deles af alle produkttyper. Underklasser som **Laptop**, **Charger**, **Monitor** og **Desktop** arver fra **Item** og udvider dets funktionalitet med specialiserede egenskaber og metoder.
5. **Automatisk ID-generering**: **Item**-klasserne implementerer automatisk ID-generering for hver produktspecifikation. Ved oprettelse tildeles hvert produkt en unik ID, hvilket sikrer, at ingen to elementer deler samme identifikator. Denne funktion forbedrer sporbarheden og letlæseligheden af produktdataene.
6. **Dokumentation og kommentarer**: Koden inkluderer omfattende kommentarer og docstrings, der giver klare forklaringer på formål, funktionalitet og brug af hver komponent. Korrekt dokumentation forbedrer læsbarheden, vedligeholdelsen og samarbejdet mellem udviklere.

I forhold til UML-diagrammerne, var de lavet i starten af projektet og skulle i reglen være opdateret men har haft fokus på koden efterfølgende.

Ud over det er der også instansieret et virtual miljø som kan bruges, i dette tilfælde er der dog ingen pakker udover basis pakker som følger med i Python blevet brugt.