Case Study I

Julian Huber & Matthias Panny

Software Architektur

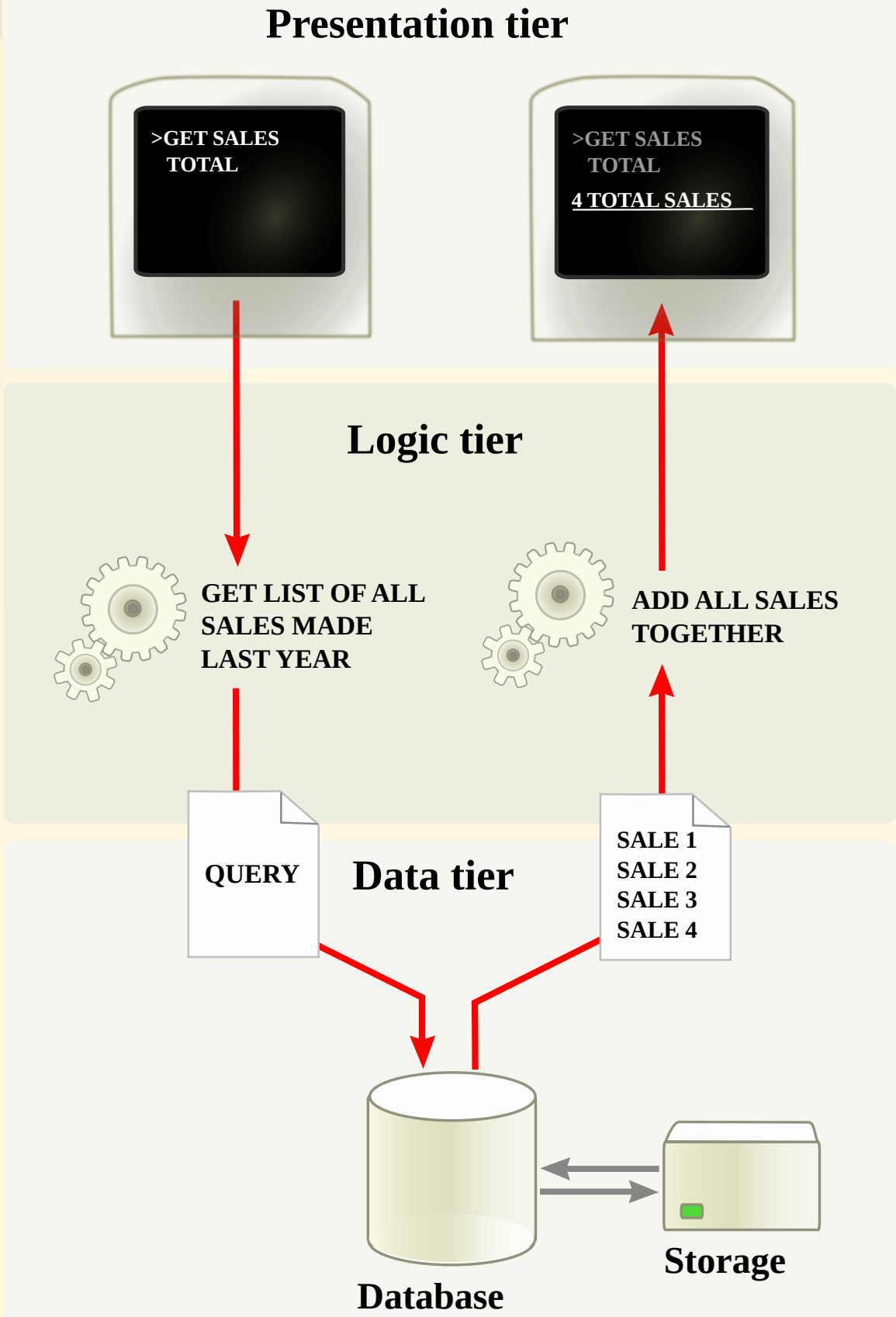
Wir wollen unsere vier Use-Cases sinnvoll strukturieren

Funktionale Anforderungen

- Das System soll über eine Weboberfläche bedient werden
- Das System soll eine Datenbank zur Speicherung der Daten verwenden
- Hierfür werden folgende Python-Module vorschlagen, welche aber nicht zwangsläufig verwendet werden müssen:
  - Datenbank: `tinydb`
  - User-Interface: `streamlit`

Funktionale Anforderungen

- Es hilft die Software in Schichtenarchitektur zu konstruieren
  1. User-Interface: Hier werden die Daten eingegeben und ausgegeben
  2. Business Logic: Hier werden die Daten verarbeitet
  3. Datenbank: Hier werden die Daten gespeichert
- → die Schichten sollen nur auf die direkt darunter liegende Schicht zugreifen können



Software Architektur - User-Interface (Frontend)

- Es bietet sich an zunächst alle Seiten des User-Interface in einem Mockup zu entwerfen
- Hierbei kann man sich an den Use Cases orientieren

Mockup des User-Interfaces

- Die Funktionsweise von `streamlit` wird in Kapitel 04_04_Case_Study_I_Streamlit näher behandelt
- Ggf. bietet es sich hier an, erst einen Prototypen mit hard-gecodeten Daten zu erstellen, bevor die Business Logic implementiert wird
- Hierfür exisitert eine unvollständige Beispiel-Implementierung `ui_device.py`

Software Architektur - Business Logic (Backend)

- Die Business Logic ist die Schnittstelle zwischen User-Interface und Datenbank
- Hier werden alle Objekte und Funktionen definiert, welche für die Verarbeitung der Daten benötigt werden
- Es bietet sich an zunächst alle Objekte und deren Methoden in einem Klassendiagramm zu entwerfen
- Ein möglicher Startpunkt für die Implementierung ist in `devices.py` bzw. `users.py` gegeben
- Sinnvoll ist es auch erst die Funktionalität der Business Logic zu testen, bevor das User-Interface und die Datenbank implementiert wird. Hierfür sind erste `unit-tests` gegeben

Software Architektur - Business Logic (Backend)

- Als Startpunkt bieten sich die oben genannten Klassen `User` und `Device` an. Allerdings kann es Sinn machen, diese Klassen noch weiter zu unterteilen
  - z.B. könnte es sinnvoll sein, die Klasse `Device` in die Klassen `Device` und `ReservationService` aufzuteilen
  - z.B. könnte es sinnvoll sein die kommenden Wartungen in einer Tabelle (DataFrame) zu speichern, welche die Geräte-ID, Namen, das Datum und die Kosten enthält. DataFrames können einfach mit `steamlit` angezeigt werden. Zum Erstellen dieses DataFrames könnte ein `Query`-Objekt erstellt werden, welches die Datenbank nach den nächsten Wartungen durchsucht und diese in einem DataFrame speichert
- vgl. 01_06_Python_Grundlagen_Objektorientierung_II

Software Architektur - Datenbank

- Die wichtigen Daten sollen in einer Datenbank gespeichert werden, damit diese auch einen Neustart des Programms überleben
- Zudem ist es sinnvoll zu Beginn einige Daten in der Datenbank zu halten, um die Funktionalität des Programms zu testen
- In diesem Fall bietet es sich an, die Datenbank direkt in der Business Logic mit entsprechende Klassen zu implementieren
- Bei der Beispiel-Implementierung wird die Datenbank `tinydb` verwendet, welche die Objekte in einer JSON-Datei speichert

Software-Stack

Software-Stack

- Wir wollen unsere Case Study I natürlich in Python implementieren
- Dazu werden wir folgende Module verwenden:
  - `streamlit` für das User Interface
  - `tinydb` für die Datenbank
- Wir arbeiten im Team → Versionskontrolle mit git & Github
- Wir bauen das gesamte Projekt in einer `venv` auf um unsere Dependencies einfach zu managen

✍ Aufgabe

- Erstellen Sie sich in Ihrem Team ein Repository auf Github & klonen Sie es auf Ihre Rechner
- Fügen Sie eine `.gitignore`-Datei für Python in Ihr Repository ein. Eine mögliche Variante ist hier zu finden.
  → commit & push
  Dies soll unbedingt vor dem Erstellen der `.venv` geschehen, da sonst die `.venv`-Datei nicht ignoriert wird
- Erstellen Sie eine `.venv` und installieren Sie die `requirements.txt`
  → commit (der `.venv`-Ordner darf nicht enthalten sein) & push
- Erstellen Sie eine `README.md`-Datei, die alle für das Projekt notwendigen Informationen und Schritte beschreibt
  → commit & push
- Erstellen Sie eine `main.py`-Datei die "Hello World" ausgibt
  → commit & push