**Parkeringsapp – Arkitekturdesign**

### **Section 1: Överblick**

Parkeringsapplikationen består av:

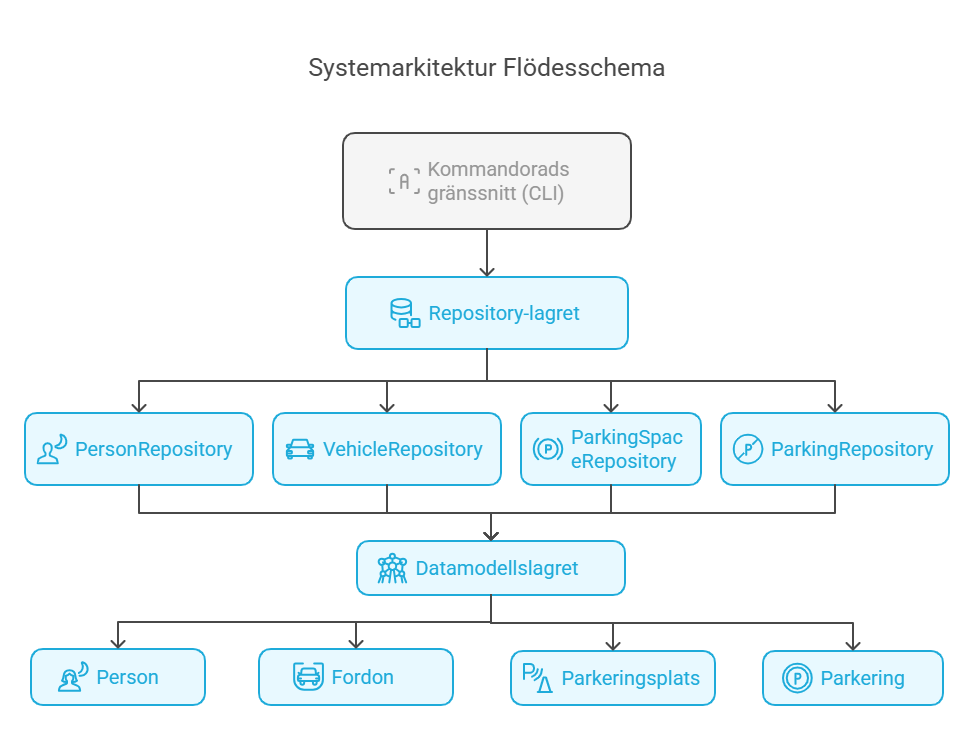
Kärndatamodeller: `Person`, `Fordon`, `Parkeringsplats` och `Parkering`.

- Datahanteringsrepository: `PersonRepository`, `VehicleRepository`, `ParkingSpaceRepository` och `ParkingRepository`.

- Kommandoradsgränssnitt (CLI): En användargränssnutt för att interagera med datamodellerna och repository.

Här är ett högnivådiagram över applikationens arkitektur:

### **Section 2: Arkitekturdiagram**



**Section 3: Detaljdesign**

#### **3.1 Datamodellslagret**

Detta lagret definierar de grundläggande entiteter i applikationen.

1. **Person**
   1. Attribut:
      1. namn (String): Namnet på personen.
      2. personnummer (String): Ett unikt identifierare för personen (10 siffror).
   2. Metoder:
      1. isValid(): Validerar att personnummer är exakt 10 siffror.
      2. toJson(): Serialiserar objektet till JSON.
      3. fromJson(): Deserialiserar JSON till ett Person-objekt.
2. **Fordon**
   1. Attribut:
      1. registreringsnummer (String): Unikt registreringsnummer för fordonet.
      2. typ (String): Typ av fordon (t.ex., "bil", "motorcykel").
      3. ägare (Person): Referens till ägaren av fordonet.
   2. Metoder:
      1. isValid(): Validerar registreringsnumret och typen.
      2. toJson(): Serialiserar objektet till JSON.
      3. fromJson(): Deserialiserar JSON till ett Fordon-objekt.
3. **Parkeringsplats**
   1. Attribut:
      1. id (int): Unikt identifierare för parkeringsplatsen.
      2. adress (String): Adressen för parkeringsplatsen.
      3. prisPerTimme (double): Priset per timme för parkering.
   2. Metoder:
      1. isValid(): Säkerställer att priset är positivt.
      2. toJson(): Serialiserar objektet till JSON.
      3. fromJson(): Deserialiserar JSON till ett Parkeringsplats-objekt.
4. **Parkering**
   1. Attribut:
      1. fordon (Fordon): Fordonet som parkerar.
      2. parkeringsplats (Parkeringsplats): Parkeringsplatsen där fordonet parkerar.
      3. starttid (DateTime): Starttid för parkeringsessionen.
      4. sluttid (DateTime?): Sluttid för parkeringsessionen (kan vara null om pågående).
   2. Metoder:
      1. beräknaKostnad(): Beräknar kostnaden för parkering baserat på varaktighet och pris per timme.
      2. toJson(): Serialiserar objektet till JSON.
      3. fromJson(): Deserialiserar JSON till ett Parkering-objekt.

#### **3.2 Repository-lagret**

Detta lagret hanterar samlingar av datamodeller.

1. **PersonRepository**
   1. Attribut:
      1. \_personer (List): Lista över alla personer.
   2. Metoder:
      1. add(Person person): Lägger till en ny person.
      2. getAll(): Returnerar alla personer.
      3. getById(String personnummer): Hämtar en person genom deras personnummer.
      4. update(Person uppdateradPerson): Uppdaterar en befintlig person.
      5. delete(String personnummer): Tar bort en person genom deras personnummer.
2. **VehicleRepository**
   1. Attribut:
      1. \_fordon (List): Lista över alla fordon.
   2. Metoder:
      1. add(Fordon fordon): Lägger till ett nytt fordon.
      2. getAll(): Returnerar alla fordon.
      3. getByRegistrationNumber(String registreringsnummer): Hämtar ett fordon genom dess registreringsnummer.
      4. update(Fordon uppdateratFordon): Uppdaterar ett befintligt fordon.
      5. delete(String registreringsnummer): Tar bort ett fordon genom dess registreringsnummer.
3. **ParkingSpaceRepository**
   1. Attribut:
      1. \_parkeringsplatser (List): Lista över alla parkeringsplatser.
   2. Metoder:
      1. add(Parkeringsplats parkeringsplats): Lägger till en ny parkeringsplats.
      2. getAll(): Returnerar alla parkeringsplatser.
      3. getById(int id): Hämtar en parkeringsplats genom dess ID.
      4. update(Parkeringsplats uppdateradParkeringsplats): Uppdaterar en befintlig parkeringsplats.
      5. delete(int id): Tar bort en parkeringsplats genom dess ID.
4. **ParkingRepository**
   1. Attribut:
      1. \_parkeringar (List): Lista över alla parkeringssessioner.
   2. Metoder:
      1. add(Parkering parkering): Lägger till en ny parkeringssession.
      2. getAll(): Returnerar alla parkeringssessioner.
      3. getByVehicleRegistrationNumber(String registreringsnummer): Hämtar en parkeringssession genom fordonets registreringsnummer.
      4. update(Parkering uppdateradParkering): Uppdaterar en befintlig parkeringssession.
      5. delete(String registreringsnummer): Tar bort en parkeringssession genom fordonets registreringsnummer.
      6. beräknaTotalInkomst(): Beräknar total inkomst från alla avslutade parkeringssessioner.

#### **3.3 Kommandoradsgränssnitt (CLI)**

CLI låter användaren interagera med applikationen.

1. **Huvudmeny**
   1. Alternativ:
      1. Hantera Personer
      2. Hantera Fordon
      3. Hantera Parkeringsplatser
      4. Hantera Parkeringar
      5. Avsluta
2. **Undermenyer** Varje undermeny erbjuder CRUD-operationer för den respektive datamodellen:
   1. Skapa
   2. Visa (Läs alla)
   3. Uppdatera
   4. Ta bort
   5. Gå tillbaka
3. **Flödesexempel**

Välkommen till Parkeringsappen! Vad vill du hantera?

1. Personer
2. Fordon
3. Parkeringsplatser
4. Parkeringar
5. Avsluta Välj ett alternativ (1-5): 1

Du har valt att hantera Personer. Vad vill du göra?

1. Skapa ny person
2. Visa alla personer
3. Uppdatera person
4. Ta bort person
5. Gå tillbaka till huvudmenyn Välj ett alternativ (1-5)

### **Section 4: Nyckelfunktioner**

1. **Validering**
   1. Säkerställer dataintegritet genom att validera indata innan bearbetning.
   2. Exempel: personnummer måste vara exakt 10 siffror.
2. **Person handler**
   1. Säkerställer att det bara finns en instans av varje repository genom att använda Singleton-mönstret.
3. **CRUD-operationer**
   1. Levererar fullständiga Create, Read, Update och Delete-funktioner för alla datamodeller.
4. **Kostnadsberäkning**
   1. Implementerar en metod i Parkering-klassen för att beräkna parkeringskostnader baserat på varaktighet och pris per timme.
5. **Sökfunktion**
   1. Tillåter sökning efter specifika objekt baserat på kriterier (t.ex., hitta alla fordon ägd av en specifik person).
6. **Serialisering/Deserialisering**
   1. Implementerar toJson() och fromJson()-metoder för framtida databasintegration.
7. **Asynkron programmering**
   1. Modifierar vissa metoder för att returnera Future-objekt för att hantera asynkrona operationer.

### **Section 5: Framtida Förbättringar**

1. **Databasintegration**
   1. Ersätt minneslagring med en permanent databas (t.ex., SQLite, Firebase).
2. **Grafiskt Gränssnitt (GUI)**
   1. Utveckla ett grafiskt gränssnitt (GUI) med hjälp av Flutter eller annan ramverk.
3. **Autentisering**
   1. Lägg till användarautentisering och auktorisering för säker åtkomst.
4. **Notifikationer**
   1. Implementera påminnelser för parkeringsutfall eller betalningsfalliga datum.
5. **Avancerad Sökning**
   1. Förbättra sökfunktionen med filter och sorteringsalternativ.