University of
South Australia

INFS 2044

Workshop 1b

Preparation Already Done

- Bring a copy of the workshop instructions (this document) to the workshop

Where We Are At

- Revisited good requirements practices
- Validated domain model and use cases

Learning Objectives

- Refine use case narratives into implementable scenarios
- Identify variability in system requirements
- Document system environment using System Context Diagrams
- Assess the impact of changes in system designs

Task 1. Use Case Completion (30 min)

- Review the use case narrative for *Make Booking*.
- Identify any relevant alternate flows that need to be considered.
- Identify any external systems that may be relevant for this use case.
- Is there any other information that has been missed?
- Is there enough information that you could write code based on what is given in the narrative?

Use Case UC01: Make Booking

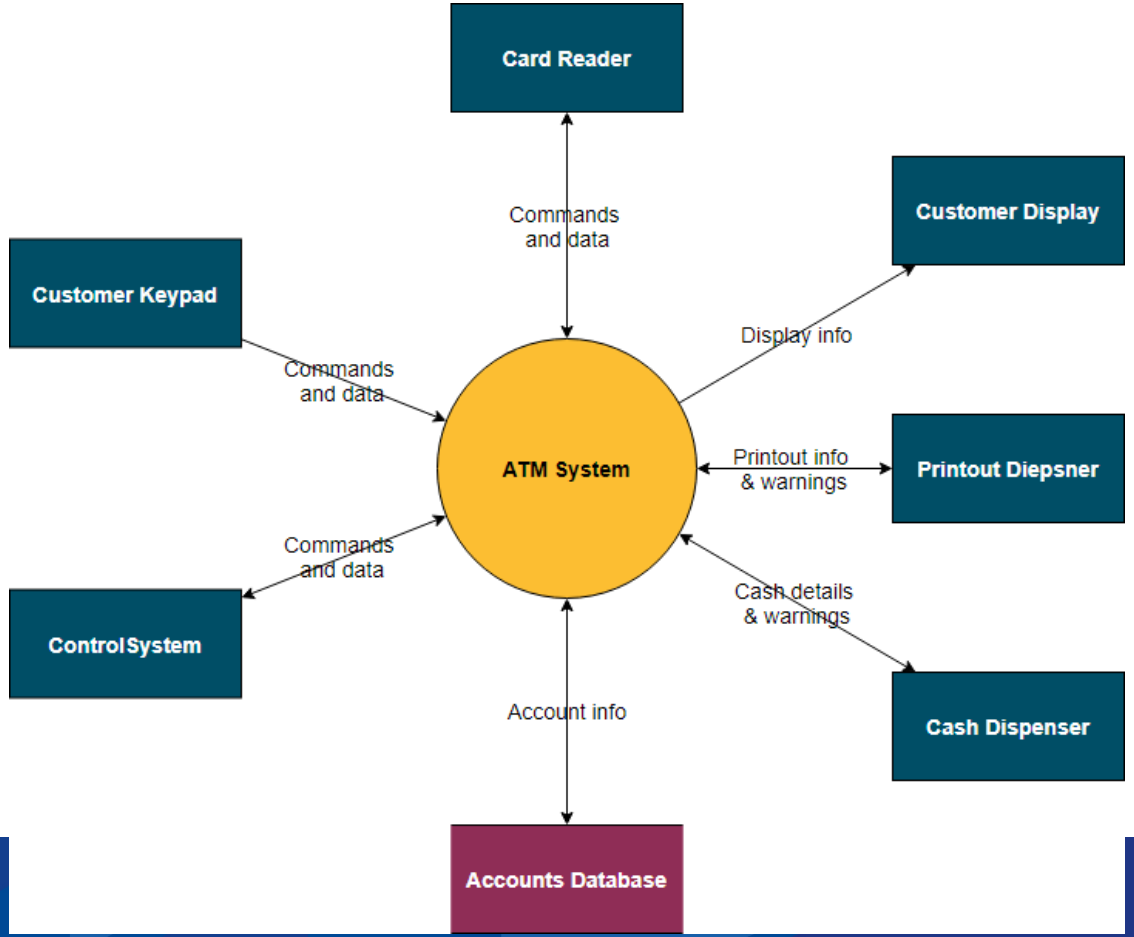
1. *User enters date range.*
2. *System presents available rooms, their descriptions, and daily rates.*
3. *User selects room.*
4. *System presents total price.*
5. *User enters contact details and payment details.*
6. *System verifies payment, records payment confirmation, and issues booking confirmation.*

Task 2: Identify Variability (20 min)

- *Identify possible variations in UC01 Make Booking*
- *What processes and technical variations could be relevant for the Booking System?*

Task 3: Context Diagram (20 min)

- A Context Diagram shows the flow of information between the system and external entities.



https://online.visual-paradigm.com/knowledge/system-context-diagram/what-is-system-context-diagram/

Draw a Context Diagram

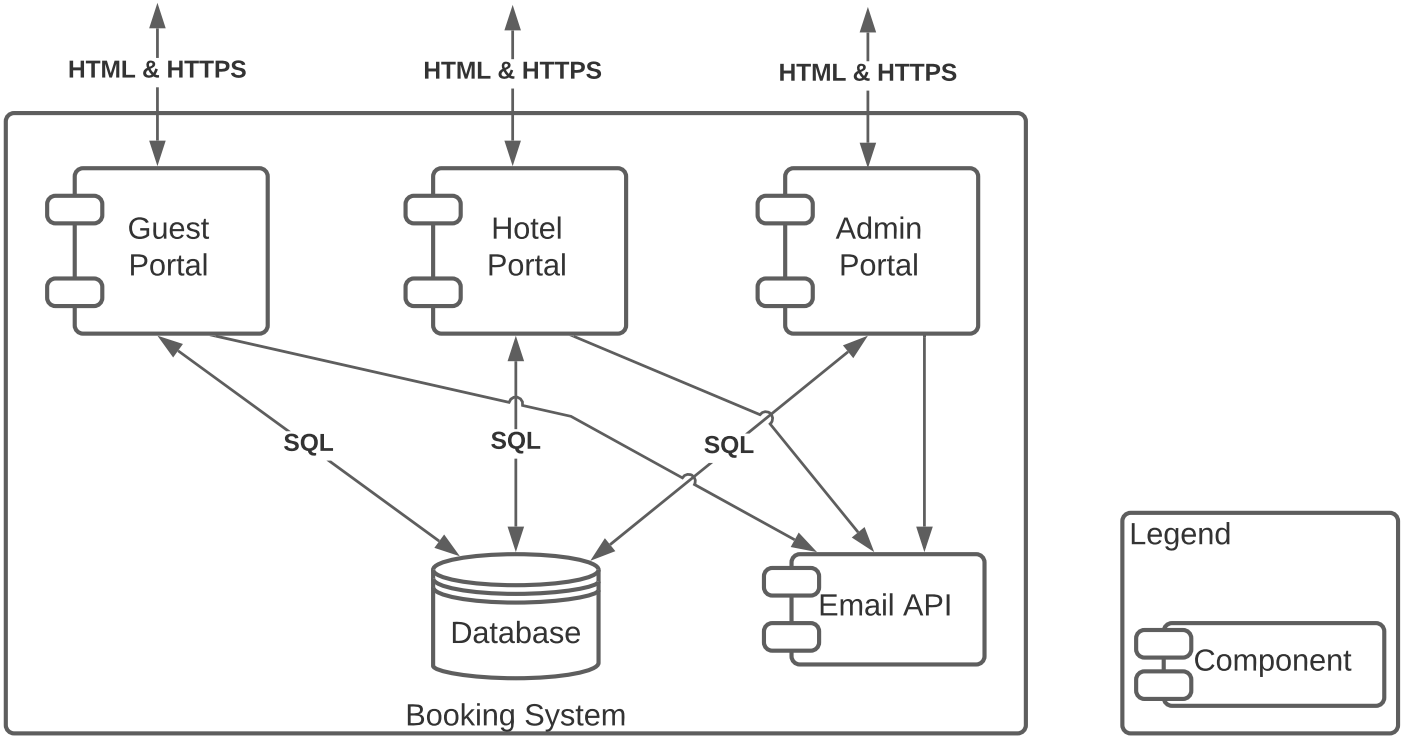
- Suppose payments are processed and verified using an external payment service such as Stripe or PayPal.
- Draw a Context Diagram for the Room Booking System example.
- Consider only *UC01 Make Booking*

Task 4: Identify Complexity (30 min)

- Design decisions are motivated by minimizing complexity and controlling the impact of changes.

- Identify changes that may be easy or difficult to make in the following system design.
- Why are some changes easy to make and others more difficult?

Booking System: Components



Booking System: Responsibilities

Component	Responsibilities
Guest Portal	Render User Interface for all Guest-accessible functions, Orchestrate Use Case logic, Validate Input, Invoke Payment Verification, Send notifications, Persist data in database
Hotel Portal	Render User Interface for all Hotel-accessible functions, Authenticate and authorize users, Orchestrate Use Case logic, Validate Input, Send notifications, Persist data in database

Booking System: Responsibilities

Component	Responsibilities
Admin Portal	Render User Interface for all Admin-accessible functions, Authenticate and authorize users, Orchestrate Use Case logic, Validate Input, Send notifications, Persist data in database
Database	Persist all data

Impact of Changes

- Are these changes easy or difficult to make in the given design?:
  - Change the database schema or change to a different database management system
  - Change the colour and layout of the Guest portal
  - Authenticate users via single-sign on, Google, and Facebook
  - Create a dedicated desktop application for hotel staff
  - Receive bookings through external booking system such as Booking.com, Wotif, etc

Impact of Changes

- Are these changes easy or difficult to make in the given design?:
  - Add a portal for booking agents who book on behalf of guests
  - Use a different payment service
  - Notify guests via Signal, WeChat, automated voice call
  - Adapt to different languages, date formats, and currency
  - Show locations using Google maps or Open Streetmap

Design Discussion

- Exposing database details to components creates undesirable dependencies
- Repeated functions in the three components causes obscurity (multiple copies of similar code that must all be updated correctly)
- Lack of defined boundaries between presentation, application logic, data storage, security, messaging, and external services creates difficult-to-maintain structures
- **Structuring the software properly is important to avoid such complexity.**

To Know

- Refine use case narratives into implementable scenarios
- Identify variability in system requirements
- Document system environment using System Context Diagrams
- Assess the impact of changes in system designs

Activities this Week

- Complete Quiz 1

University of
South Australia