- The name of our project:

"*Logic-based Travel planner*"

ID	Name	Group
100308	Hosam Khaled Bahnasy Dyab	G3-CS
101537	Hazem Mohamed Zaki Labib	G3-CS
102747	Khaled Gamal Saber Fouda	G3-CS
100614	Basmala Ayman Abdelhalim Mohamed	G٢-CS
99690	Aya Mohamed Salah Eldin	G2-CS
101602	Anas Magdy Saied Elkhafif	G2-CS
102086	Abdulrahman Ghandour Mohammed	CASE

Supervised by:

Prof. Dr. Hesham ElDeeb

T.A. Nardin Wagih

Table of Contents

1. Problem Definition

Before we had *logic-based systems* like **Prolog** to help with travel planning, it wasn't easy for travelers to get their trips organized efficiently. People often struggled to find *trustworthy recommendations* for **destinations**, had a tough time trying to **accurately budget**, and couldn't easily gather **info** about cities, like *landmarks*, *tourist spots*, and *cultural attractions*. When they finally arrived at their destination, they faced even more challenges, like picking **budget-friendly hotels**, figuring out **transportation options**, and deciding on fun **activities** such as *shopping*, catching a film at the *cinema*, or going on *museum tours*. All these obstacles made travel planning a *time-consuming*, *confusing*, and often *frustrating* process. The **Logic-based Travel Planner** steps in to tackle these problems by offering an *automated*, **logic-driven framework** that makes decision-making easier, optimizes **how resources are used**, and really enhances the travel experience.

2. Project Description

2.1 Introduction

The **Logic-based Travel Planner** is a cool *software tool* that aims to change the way we plan our trips by giving you **customized**, *personalized travel itineraries*. Built as a key project for the *Logic Programming course*, this system makes use of **Prolog's declarative programming model** to make the planning process *smooth* and *well-organized*. It helps travelers by **calculating budgets** based on how long they'll be away (whether it's days or weeks), **recommending hotels** that fit their *financial limits*, and pointing out **top-rated places to stay** in the city they're visiting. Users can pick from **various cities or countries** and get **in-depth info** about *things to see*—from *iconic landmarks* to *tourist hotspots* and *cultural spots*—while also getting help with **transportation** and **custom suggestions** for *hotels*, *attractions*, and *activities* that match their preferences. And it's not just about logistics; the system makes your trip more enjoyable by helping you discover **fun activities**, like *shopping*, *movie outings*, and *museum visits*, so you can skip tedious *manual research*.

2.2 Technical Approach

The **Logic-based Travel Planner** is architected as a **rule-based expert system**, leveraging **Prolog's logic programming capabilities** to address the *constraint-driven complexity* of travel planning. This section outlines the system's components, design principles, and implementation details with a focus on **precision** and **scalability**.

2.2.1 System Components

1. **Knowledge Base**:
    - **Facts**: Structured predicates encapsulate critical travel data:
        - **city/1:** Defines **available cities**
          (e.g., cairo, aswan, portsaid).
        - **attraction/2:** Enumerates **attractions per city**
          (e.g., pyramids, museums).

- **hotel/4:** Specifies hotels with four arguments—**City**, **Name**, **PricePerNight**, **Rating**—capturing *location*, *identity*, *cost*, and *quality* *(e.g., hotel(cairo, 'four_seasons', 2000, 4.8)).*

- **transport/3:** Lists **transportation modes** with *costs* *(e.g., transport(cairo, car, 500)).*

- **activity/3:** Defines **activities** with *costs* *(e.g., activity(cairo, shopping, 300)).*

2. **Scalability**: The knowledge base is *modular*, supporting future additions like *seasonal pricing* or *user reviews*.

2.2.3 Technical Features

1. **Budget Estimation:**
   - Computes a **comprehensive budget** by aggregating:
     - **Accommodation:** PricePerNight * Days from hotel/4.
     - **Transportation:** Fixed cost from transport/3.
     - **Food:** Daily estimate (e.g., 300 EGP/day).
     - **Activities:** Sum of selected activity costs from activity/3.

♦ **Example:** For a **3-day trip** to *Cairo*, it calculates PricePerNight * Days + TransportCost + (FoodCost * Days).

2. **Hotel Recommendations:**
   - Filters hotel/4 based on **PricePerNight** to recommend options within *budget*.
   - Suggests **top-rated hotels** using Rating (*e.g., ≥ 4.2*), implemented via **top_rated_hotel/2**.
   - Leverages the ***/4 arity*** to balance *cost* and *quality* considerations.

3. **Destination Information:**
   - Queries attraction/2 to provide **detailed insights** into *landmarks*, *tourist attractions*, and *cultural sites* for selected cities.

- Supports *multi-destination queries* for comparative planning.

4. **Transportation Assistance:**
   - Matches transport/3 options to cities, optimizing for *cost* and *user preference*.
   - Future enhancements will incorporate *distance-based cost models*.
5. **Personalized Recommendations:**
   - Implements **personalized_recommendation/4** to align *hotels*, *attractions*, and *activities* with **user preferences**, ensuring *budget compliance* and *preference matching*.
6. **Activity Discovery:**
   - Maps activity/3 to **user interests** (e.g., *shopping*, *cinema*, *museums*), enhancing *leisure planning* with *cost-aware suggestions*.

2.2.4 Design Principles

- **Modularity:** *Facts* and *rules* are independent, enabling updates without redesign.
- **Efficiency: Prolog's backtracking** ensures *polynomial-time performance* for mid-term dataset size.
- **Scalability:** Supports incremental enhancements (*e.g., real-time data, multi-user support*).
- **Reliability:** *Input validation* and *constraint checking* ensure robust outputs.

---

3. Diagrams

3.1 Use Case Diagram

The Use Case Diagram for the **Logic-based Travel Planner** (illustrated in **Figure 1**) provides a visual representation of the functional requirements and interactions between the system's actors and its core processes. This diagram outlines the primary functionalities offered to the **User** and the supporting roles of the **System** and **Knowledge Base** actors, ensuring comprehensive travel planning experience. Below is a detailed explanation of the diagram's components and their relationships.

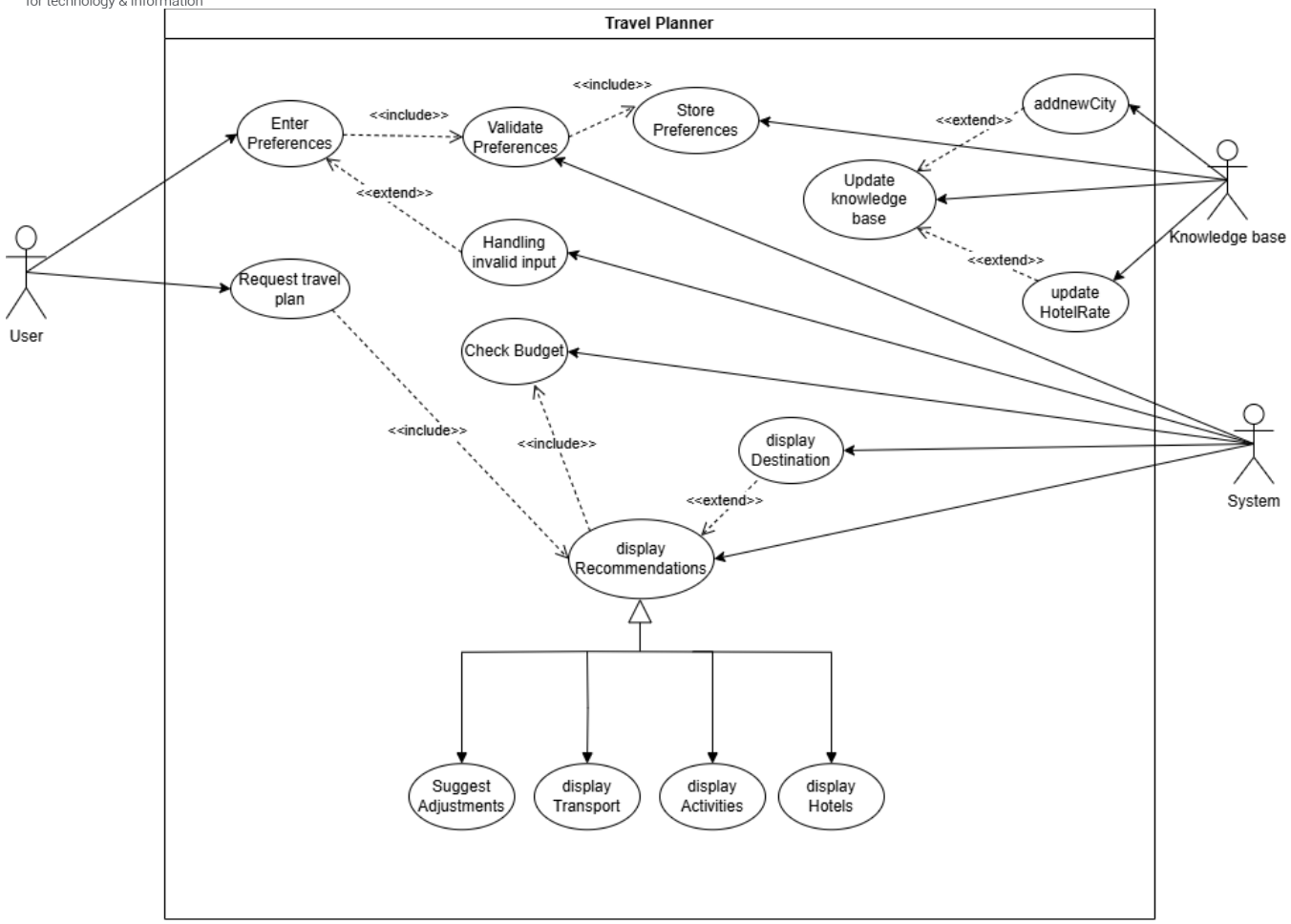

Figure 1: Use case diagram

3.1.1 Actors

- **User**: Represents the end-user who interacts with the system to plan their travel. The **User** initiates all primary actions, such as entering preferences and requesting travel plans.
- **System**: Acts as the operational backbone, executing processes like validation, recommendation generation, and knowledge base updates in response to user inputs.
- **Knowledge Base**: Serves as a dynamic repository that stores and updates travel-related data (e.g., cities, hotels, attractions), interacting with the **System** to provide and refine information.

3.1.2 Primary Use Cases

1. **Enter Preferences**:
    - The **User** begins by entering their travel preferences (e.g., trip type, budget, duration), which is a foundational step in the planning process. This use case includes:
        - **Validate Preferences**: The **System** validates the entered preferences (e.g., checking budget feasibility) to ensure they are actionable.
        - **Store Preferences**: Validated preferences are stored in the **Knowledge Base** for later use in generating travel plans.
    - **Relationship**: The <<include>> relationship indicates that validation and storage are mandatory subprocesses of entering preferences.

2. **Request Travel Plan**:
    - The **User** requests a tailored travel plan based on their stored preferences. This triggers a series of dependent actions:
        - **display Destination**: The **System** provides information about potential destinations.
        - **Show Recommendations**: The core outcome, displaying a comprehensive plan including hotels, transport, and activities.
    - **Relationship**: The <<include>> relationship ensures that destination display and recommendations are integral to the request process.
3. **Show Recommendations**:
    - This central use case generates and presents the travel plan, incorporating:
        - **Check Budget**: The **System** verifies that the plan aligns with the user's budget.
        - **display Transport**: Suggests transportation options.
        - **display Activities**: Lists available activities (e.g., shopping, museums).
        - **display Hotels**: Recommends suitable accommodations.
    - **Extension**: **Suggest Adjustments** (<<extend>>) is triggered if the budget is invalid, allowing the **User** to modify their preferences.
    - **Relationship**: The <<include>> relationships highlight the multi-faceted nature of recommendations, while the extension provides flexibility.

4. **Update Knowledge Base**:
    - The **System** updates the **Knowledge Base** to reflect new data or changes, including:
        - **addnewCity** (<<extend>>): Adds a new city (e.g., Luxor) to expand the system's scope.
        - **updateHotelRate** (<<extend>>): Adjusts hotel ratings or prices (e.g., updating Four Seasons to 2200 EGP).
    - **Relationship**: The <<extend>> relationships indicate optional enhancements to the knowledge base.

3.1.3 Extended Use Cases

- **Handling Invalid Input** (<<extend>> from **Enter Preferences**):
    - If the **User** provides invalid preferences (e.g., insufficient budget), the **System** handles the input error, prompting the **User** to re-enter data.
- **Suggest Adjustments** (<<extend>> from **Show Recommendations**):
    - Activated when the **Check Budget** process identifies an invalid budget, offering the **User** suggestions to adjust their plan.

3.2 Sequence Diagram

The Sequence Diagram for the **Logic-based Travel Planner** (illustrated in **Figure 2**) depicts the dynamic interaction between the **User**, **System**, **Budget System**, and **Knowledge Base** actors over time. It illustrates the step-by-step process of entering preferences, generating a travel plan, checking budget constraints, and updating the knowledge base, with activation bars highlighting active processing periods. Below is a detailed explanation of the diagram's flow and its significance.

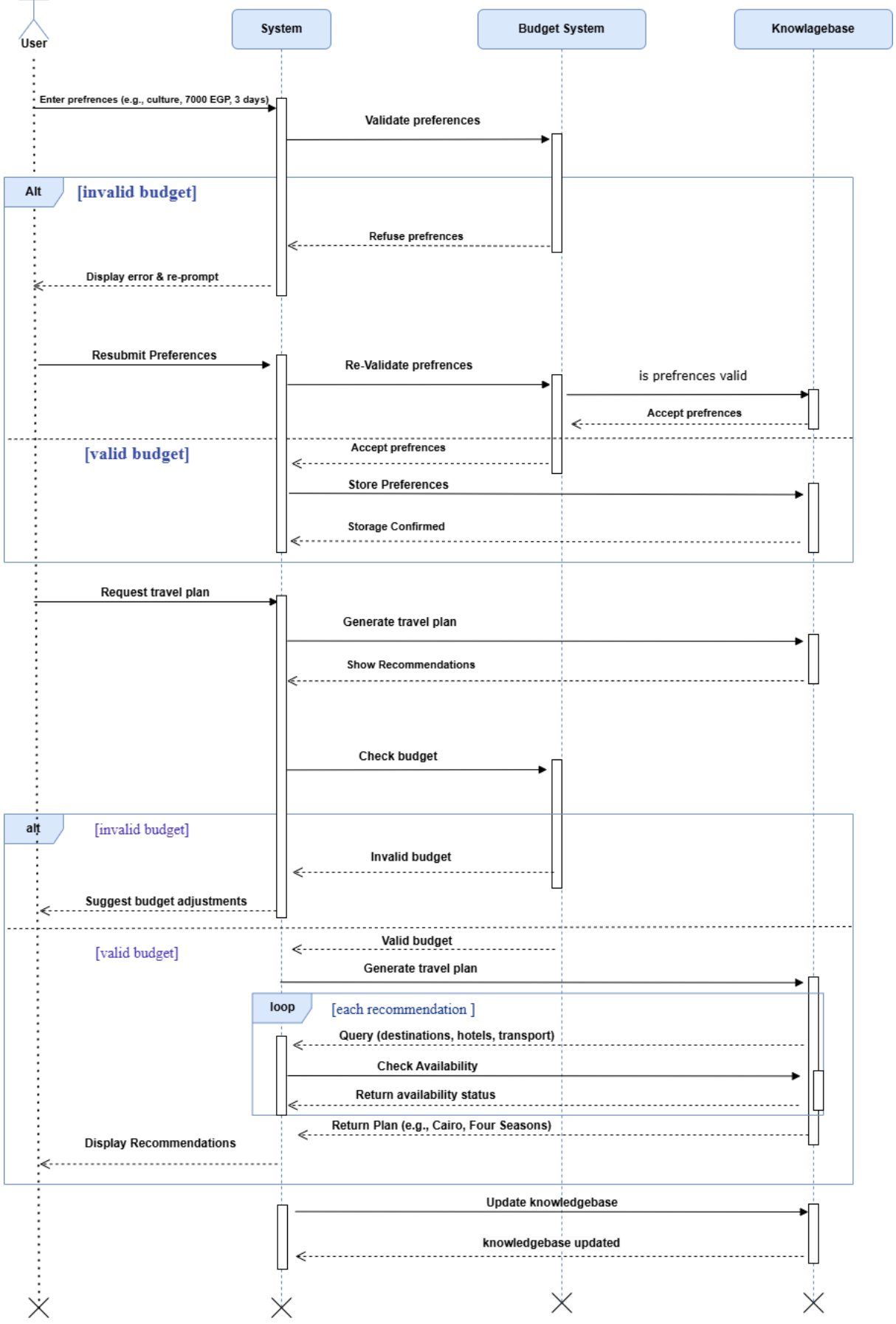

Figure 2:Sequence diagram

3.2.1 Actors and Participants

- **User**: The initiator of the process, sending requests and receiving responses.
- **System**: The central coordinator, managing interactions with the **Budget System** and **Knowledge Base**.
- **Budget System**: A specialized component responsible for validating and checking budget constraints.
- **Knowledge Base**: A data repository that stores and retrieves travel-related information (e.g., cities, hotels, preferences).

3.2.2 Sequence Flow

1. **Entering Preferences**:
    - The **User** sends a message **"Enter preferences (e.g., culture, 7000 EGP, 3 days)"** to the **System**.
    - The **System** activates and forwards the request to the **Budget System** with **"Validate preferences"**.
    - **Alternative Flow ([Invalid budget])**:
        - If the budget is invalid, the **Budget System** returns **"Refuse preferences"** to the **System**.
        - The **System** responds to the **User** with **"Display error & re-prompt"**, prompting resubmission.
        - The **User** resubmits preferences (**"Resubmit Preferences"**), and the **System** re-engages the **Budget System** with **"Re-validate preferences"**.
        - If still invalid, the process exits with **"Log error & exit"**; otherwise, it proceeds.
    - **Main Flow ([Valid budget])**:
        - If valid, the **Budget System** sends **"Accept preferences"** to the **System**.
        - The **System** instructs the **Knowledge Base** to **"Store preferences"**, receiving **"Storage confirmed"** in return.
2. **Requesting Travel Plan**:
    - The **User** sends **"Request travel plan"** to the **System**.
    - The **System** retrieves data from the **Knowledge Base** with **"Retrieve preferences from KnowledgeBase"**, receiving the stored preferences.
    - The **System** then commands the **Knowledge Base** to **"Generate travel plan"**, which returns **"Show Recommendations"**.
3. **Budget Checking and Plan Generation**:
    - The **System** sends **"Check budget"** to the **Budget System**.
    - **Alternative Flow ([Invalid budget])**:
        - If invalid, the **Budget System** returns **"Invalid budget"**, and the **System** suggests **"Suggest budget adjustments"** to the **User**.
    - **Main Flow ([Valid budget])**:
        - If valid, the **Budget System** returns **"Valid budget"**, and the **System** initiates a **loop** for each recommendation:

        - The **Knowledge Base** queries **"Query (destinations, hotels, transport)"**.
        - The **System** checks **"Check availability"** with the **Knowledge Base**, which responds with **"Return availability status"**.
        - The loop concludes with the **Knowledge Base** returning **"Return plan (e.g., Cairo, Four Seasons)"**.
    - The **System** then sends **"Display recommendations"** to the **User**.
4. **Updating Knowledge Base**:
    - The **System** sends **"Update knowledgebase"** to the **Knowledge Base**.
    - **Alternative Flow**:
        - If a **new city** is added (**"Add new city (e.g., Luxor)"**) or a **hotel rate** is updated (**"Update hotel rate (e.g., Four Seasons to 2200)"**), the **Knowledge Base** processes accordingly.
    - The **Knowledge Base** confirms with **"Knowledgebase updated"**, and the **System** deactivates.

3.2.3 Control Structures

- **Alt ([Invalid budget])**: Handles budget validation with two branches—re-prompting the **User** or exiting if unresolved.
- **Loop ([Each recommendation])**: Iterates over destination, hotel, and transport queries to compile the travel plan.
- Activation bars indicate the duration of processing for each actor, with solid lines for requests and dashed lines for responses.

3.3 Activity Diagram

The Activity Diagram for the **Logic-based Travel Planner** (illustrated in **Figure 3**) models the procedural workflow of the system, from entering user preferences to generating and displaying a travel plan, including budget validation and knowledge base updates. It captures the sequence of activities, decision points, loops, and conditional paths, providing a comprehensive view of the system's operational logic. Below is a detailed explanation of the diagram's flow and its significance.

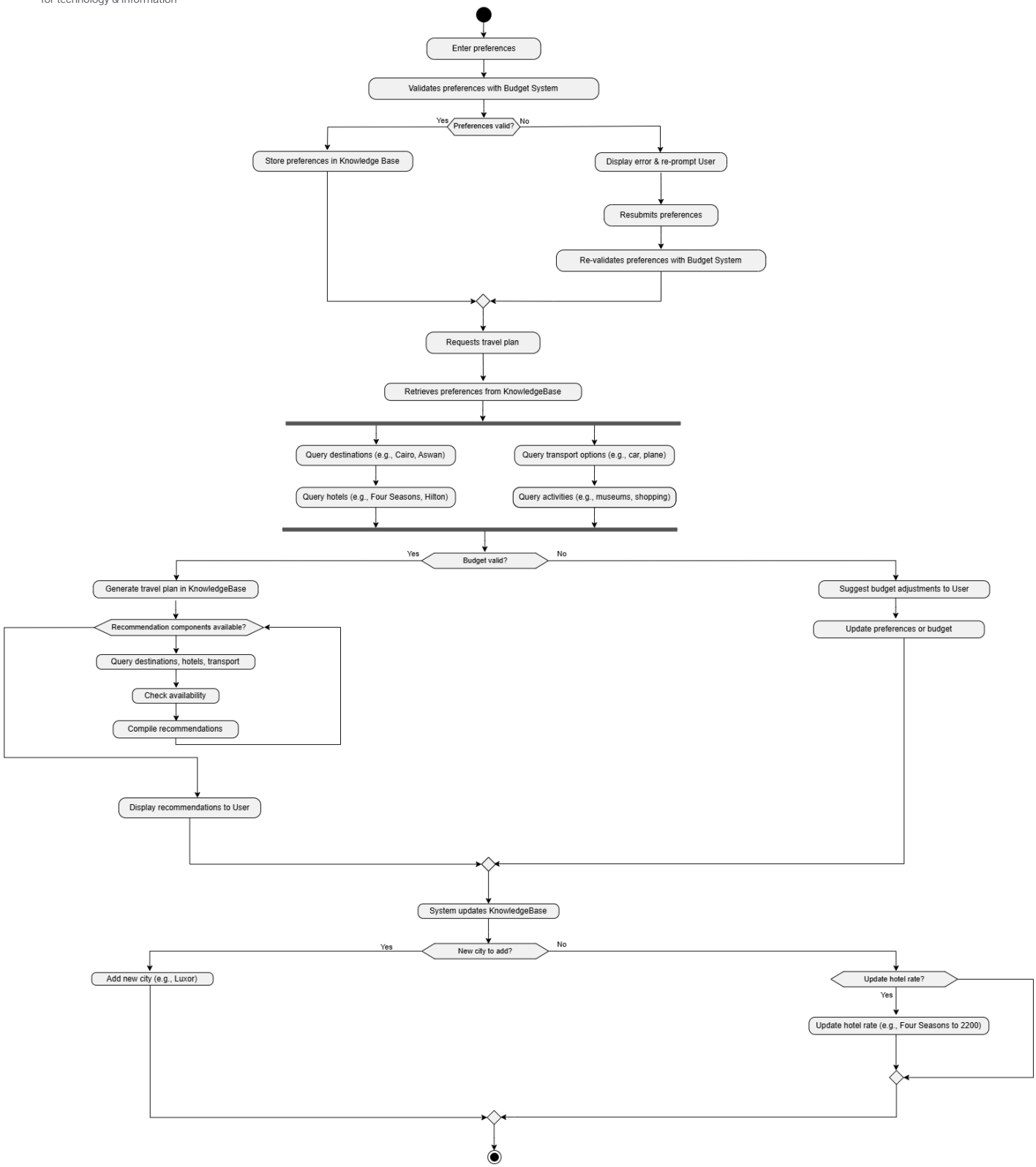

Figure 3: Activity diagram

3.3.1 Workflow Overview

The diagram kicks off with a **start node** and wraps up with a **stop node**, illustrating the full journey of travel planning. It brings together user interactions, system processes, and updates to the knowledge base, featuring decision points to tackle validation and budget limits.

3.3.2 Activity Flow

1. **Enter Preferences**:
    - The process starts with the activity **"Enter preferences in System"**, where the **User** inputs preferences (e.g., trip type, budget, duration).
    - This leads to **"Validate preferences with Budget System"**, a decision point:
        - **Yes (Preferences Valid)**: Proceeds to **"Store preferences in KnowledgeBase"**, saving the data for later use.
        - **No (Preferences Invalid)**: Triggers **"Display error with Budget System"**, followed by **"Re-prompt User"** and **"Resubmit preferences"**.
            - A nested decision **"Re-validate preferences with Budget System"** checks the resubmitted data:
                - **Yes (Valid)**: Continues to**"Store preferences in KnowledgeBase"**.
                - **No (Still Invalid)**: Exits with **"Log error & exit"**, ending the process.
2. **Request Travel Plan**:
    - After storing preferences, the **User** initiates **"Request travel plan"**.
    - The **System** performs **"Retrieve preferences from KnowledgeBase"** to access the stored data.
    - This is followed by **"Generate travel plan in KnowledgeBase"**, initiating the core planning process.
3. **Generate Travel Plan**:
    - The **Knowledge Base** executes a series of queries:
        - **"Query destinations (e.g., Cairo, Aswan)"**, **"Query hotels (e.g., Four Seasons, Hilton)"**, **"Query transport"**, and **"Query activities (e.g., museums, shopping)"**.
    - These queries feed into **"Generate travel plan in KnowledgeBase"**, which consolidates the data.
4. **Budget Validation**:
    - A decision point **"Budget valid?"** evaluates the generated plan:
        - **Yes (Valid Budget)**: Proceeds to a **loop** for recommendation compilation:
            - **"Query (destinations, hotels, transport)"**: Collects data for each component.
            - **"Check availability"**: Verifies the feasibility of the selections.
            - The loop iterates until all components are processed, then moves to **"Compile recommendations"** and **"Display recommendations to User"**.

- **No (Invalid Budget)**: Triggers **"Suggest budget adjustments to User"**, followed by **"Update preferences or budget"**, looping back to **"Generate travel plan in KnowledgeBase"** to refine the plan.

5. **Update Knowledge Base**:
   - The **System** initiates **"System updates KnowledgeBase"**, followed by a decision:
     - **"New city added?"**:
       - **Yes**: Executes **"Add new city (e.g., Luxor)"**.
       - **No**: Proceeds to **"Update hotel rate?"**:
         - **Yes**: Performs **"Update hotel rate (e.g., Four Seasons to 2200)"**.
         - **No**: Skips further updates.
   - The process concludes with **"KnowledgeBase updated"**, ensuring the system remains current.
6. **Finalization**:
   - The workflow ends with the final node, indicating completion of the travel planning process.

3.3.3 Control Structures

- **Decision Points**:
  - **"Validate preferences with Budget System?"** and **"Re-validate preferences with Budget System?"** ensure robust input validation.
  - **"Budget valid?"** handles budget constraints, offering a feedback loop for adjustments.
  - **"New city added?"** and **"Update hotel rate?"** manage knowledge base updates dynamically.
- **Loop**: The **"Query (destinations, hotels, transport)"** loop ensures all components of the travel plan are processed iteratively, enhancing *efficiency*.
- **Swimlanes**: The diagram implicitly uses swimlanes to separate responsibilities between the **System**, **User**, and **Knowledge Base**, though not explicitly labeled.

---

4. Conclusion

The **Logic-based Travel Planner** is a *big step forward* in using **logic programming** to tackle the many challenges of travel planning. This mid-term submission delivers a *strong*, **rule-based system** that combines **budget estimation**, **hotel recommendations** (through the hotel/4 predicate), **destination details**, **transportation support**, and **activity discovery** within a **Prolog framework**. The system's main features—shown in the **Use Case Diagram**, **Sequence Diagram**, and **Activity Diagram**—emphasize a *user-friendly design* that guarantees *customized, budget-friendly travel plans*. By confirming user preferences, creating personalized suggestions, and updating the **Knowledge Base** in real time, the system makes the travel planning process smoother and turns a traditionally *time-consuming* task into an *efficient, knowledgeable experience*.

The way we've set things up uses **Prolog's smart reasoning abilities** along with a modular *knowledge base*, which gives us a great foundation for scaling things up and ensuring reliability. The introduction of rules like estimated_budget/3 and personalized_recommendation/4 shows how the system can handle complex limits, while the **CLI interface** gives a practical platform for testing and validation. The diagrams show how flexible the system really is. They help with everything from handling incorrect inputs to suggesting budget adjustments, making sure that users have a *hassle-free experience*.

Looking ahead, the **Logic-based Travel Planner** is ready for more upgrades in the final phase. Future versions will enhance the **Knowledge Base** with *real-time data* (such as seasonal prices, user feedback), improve rules for *multi-day itineraries*, and add a **graphical user interface (GUI)** to boost *accessibility*. New features, like allowing *multiple users* and better activity scheduling, are going to make the system even more useful. This project not only meets the goals of the *Logic Programming course* but also demonstrates the *real-world application* of **Prolog** in solving practical problems, paving the way for future advancements in *intelligent travel planning systems*.

5. References

1. Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *The Unified Modeling Language User Guide* (2nd ed.). Addison-Wesley.

2. Bratko, I. (2012). *Prolog Programming for Artificial Intelligence* (4th ed.). Addison-Wesley.

3. Object Management Group (OMG). (2023). *Unified Modeling Language (UML) Specification, Version 2.5.1*. Retrieved from https://www.omg.org/spec/UML/2.5.1

4. SWI-Prolog. (2023). *SWI-Prolog Documentation*. Retrieved from https://www.swi-prolog.org/

5. Sommerville, I. (2015). *Software Engineering* (10th ed.). Pearson.

6. Prof. Dr. Hesham El-Deeb's book