System Requirements Specification Index

For

Ecosystem Simulation System

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

Ecosystem Simulation System
System Requirements Specification

1 Project Abstract

EcoLearn Foundation, a non-profit dedicated to environmental education, requires an interactive Ecosystem Simulation program to promote ecological awareness among students and the general public. The simulation will model interactions between different organisms in a virtual environment, demonstrating key ecological concepts such as food chains and environmental adaptation. By visualizing these relationships, EcoLearn aims to foster a deeper understanding of ecosystem dynamics and the importance of biodiversity conservation.

2 Business Requirements:

Screen Name	Console input screen
Problem Statement	1. System needs to simulate different types of organisms (plants, herbivores, carnivores) 2. System must support basic ecosystem operations such as organism creation and interaction 3. Console implementation must demonstrate object-oriented programming concepts like Classes and objects, Inheritance and polymorphism

3 Constraints

3.1 Class and Method Requirements

1. `ReservationException` Class:
    - Base exception for the entire system

- Methods
  - `__init__(message, error_code=None)`: Initialize with message and optional error code
  - `__str__()`: Return formatted error message with error code if available

2. Required Exception Subclasses:
   - `CourtUnavailableError(court_id, time_slot)`: For unavailable courts
   - `PaymentFailedError(reservation_id, amount)`: For payment failures
3. `Court` Class
   - Attributes:
     - `court_id`: String identifier for the court
     - `hourly_rate`: Float price per hour
     - `schedule`: Dictionary mapping dates to reserved time slots
   - Methods:
     - `__init__(court_id, hourly_rate)`: Initialize court with try-except-else pattern
     - `is_available(date, time_slot)`: Check availability with try-except-else pattern

4. `Reservation` Class
   - Attributes:
     - `reservation_id`: Unique identifier for the reservation
     - `player_name`: Name of the player making the reservation
     - `court`: Reference to the Court object
     - `date`: Date of reservation
     - `time_slot`: Time slot of reservation
     - `status`: String status of reservation
     - `total_cost`: Float total cost of the reservation
   - Methods:
     - `__init__(reservation_id, player_name, court, date, time_slot)`: Initialize with try-except-else pattern
     - `process_payment(payment_method)`: Process payment with try-except-else-finally pattern
5. `ReservationSystem` Class

- Attributes:
  - `courts`: Dictionary mapping court IDs to Court objects
  - `reservations`: Dictionary mapping reservation IDs to Reservation objects
  - `transaction_log`: List of dictionaries containing transaction records
- Methods:
  - `__init__()`: Initialize empty reservation system
  - `add_court(court_id, hourly_rate)`: Add court with try-except-else-finally pattern
  - `make_reservation(reservation_id, player_name, court_id, date, time_slot)`: Make reservation with try-except-else-finally pattern
  - `cancel_reservation(reservation_id)`: Cancel reservation with try-except-else-finally pattern
  - `_rollback_cancellation(reservation)`: Roll back cancellation with try-except-finally pattern

3.2 Function Constraints

1. `generate_report(system)`:
   - Plants generate energy through photosynthesis based on weather
     - Generate usage report with try-except-else-finally pattern
     - Return dictionary with report data
     - Must use all four exception handling components
   - `main()`:
     - Demonstrate all exception handling patterns
     - Must include examples of all error types
     - Must show proper usage of try-except-else-finally blocks

3.3 Error Handling Constraints

1. `try` Block Usage:
   - All operations that could raise exceptions must be in try blocks
   - Code should be structured to isolate potential error sources
2. `except` Block Usage:
   - Must catch specific exceptions, not generic Exception where possible
   - Handle exceptions appropriately with meaningful error messages

- Re-raise exceptions when appropriate

3. `else` Block Usage:
   - Must be used for code that should run only when no exceptions occur
   - Use for transaction completion and state updates
4. `finally` Block Usage:
   - Must be used for cleanup operations that always need to happen
   - Handle rollback operations when required
   - Required in methods with state changes

3.4 Data Constraints

1. Transaction Log Structure:
   - Each transaction must include:
     - `type`: String type of transaction
     - `status`: String status ("pending", "completed", "failed")
     - `error`: String error message if failed (optional)
2. Report Structure:
   - Must include date
   - Must include court statistics
   - Must include reservation counts
   - Must include revenue information

3.5 Implementation Constraints

1. No bare except blocks allowed
2. All transaction operations must use the complete try-except-else-finally pattern
3. All transactions must have rollback capability in case of errors

4. Template code Structure:

1. Exception classes

- Base ReservationException
- Specialized exception subclasses

2. Court Class:

- Court properties with validation
- Availability checking with error handling

3. Reservation Class:

- Reservation creation with validation
- Payment processing with error handling

4. ReservationSystem class

- Court management
- Reservation processing
- Transaction logging

5. Utility functions

- Report generation

6. Main Program

- Demonstration of all exception handling patterns

5. EXECUTION STEPS TO FOLLOW:

1. Create exception hierarchy
2. Implement Court class with validation
3. Develop Reservation class with error handling
4. Build ReservationSystem class with transaction integrity
5. Implement report generation function
6. Create main function demonstrating all patterns