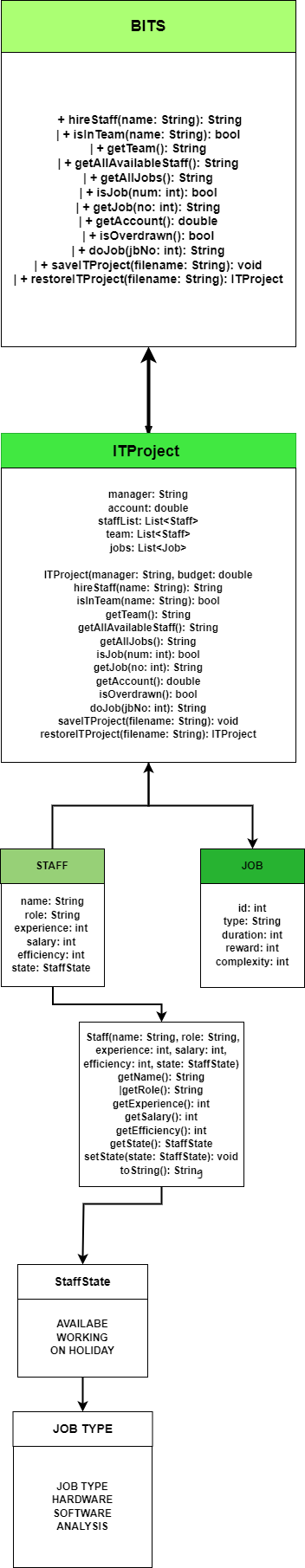
**Technical Report for ITProject Implementation**

**UML Diagram**

****

**Design Decisions**

**1. Class Structure and Relationships**

**Design Decision:** We decided to create three main classes: ITProject, Staff, and Job. Additionally, we used enums StaffState and JobType to handle staff states and job types, respectively. The ITProject class implements the BITS interface and manages the main functionalities of the project simulation.

**Alternative Implementations:**

1. **Single Class Implementation:** All functionalities could be implemented within the ITProject class without separate classes for Staff and Job.
   * **Pros:**
     + Simpler structure with fewer files.
   * **Cons:**
     + Code becomes complex and harder to maintain.
     + Low cohesion and high coupling, making it difficult to manage individual responsibilities.
2. **Separate Manager Class:** Introduce a separate Manager class to handle the project manager's responsibilities and interactions with the ITProject class.
   * **Pros:**
     + Clear separation of responsibilities.
     + Better code organization.
   * **Cons:**
     + Additional complexity in class interactions.
     + Potential for increased overhead in managing multiple classes.

**Final Decision:** We chose the structure with ITProject, Staff, and Job classes. This decision promotes high cohesion by encapsulating related functionalities within appropriate classes, leading to a more modular and maintainable design. The use of enums for staff states and job types simplifies state management and improves code readability.

**2. Data Storage for Staff and Jobs**

**Design Decision:** We used ArrayList for storing staff and job objects within the ITProject class. This choice was made to simplify the management of dynamic lists of objects.

**Alternative Implementations:**

1. **HashMap Implementation:** Use HashMap for storing staff and jobs, with staff names and job IDs as keys.
   * **Pros:**
     + Faster lookups and retrievals.
     + Efficient management of unique identifiers.
   * **Cons:**
     + Increased complexity in managing data structures.
     + Overhead of managing key-value pairs.
2. **LinkedList Implementation:** Use LinkedList for storing staff and jobs.
   * **Pros:**
     + Better performance for frequent insertions and deletions.
   * **Cons:**
     + Slower random access times.
     + More complex node management.

**Final Decision:** We chose ArrayList due to its simplicity and efficiency in managing dynamic lists. While HashMap offers faster lookups, the use case did not require such optimization, and the simplicity of ArrayList provided a balance between performance and maintainability.

**3. Method Implementation and Error Handling**

**Design Decision:** We implemented methods such as hireStaff, doJob, and saveITProject within the ITProject class. Each method includes basic error handling and returns meaningful messages for different scenarios.

**Alternative Implementations:**

1. **Exception Handling:** Use Java exceptions for managing errors and invalid operations.
   * **Pros:**
     + Clear and consistent error management.
     + Ability to separate normal flow from error handling.
   * **Cons:**
     + Increased complexity in method implementation.
     + Overhead of defining and managing custom exceptions.
2. **Return Codes:** Use integer return codes to indicate success or failure of operations.
   * **Pros:**
     + Simple implementation.
     + Easy to integrate into existing methods.
   * **Cons:**
     + Less readable and harder to maintain.
     + Requires additional documentation to interpret return codes.

**Final Decision:** We opted for a mix of meaningful return messages and basic error handling within methods. This approach balances simplicity and readability, making it easier for users to understand the outcomes of operations. While exception handling offers robust error management, the project requirements did not necessitate such complexity.

**Conclusion**

The design and implementation of the ITProject system followed principles of modularity, maintainability, and simplicity. By creating separate classes for Staff and Job and using ArrayList for data storage, we achieved a balance between performance and ease of maintenance. The method implementations include basic error handling, providing meaningful feedback to users. These design decisions collectively contribute to a well-structured and functional project simulation.