

Travel Reimbursement System, Graphical Database design and queries using Neo4j

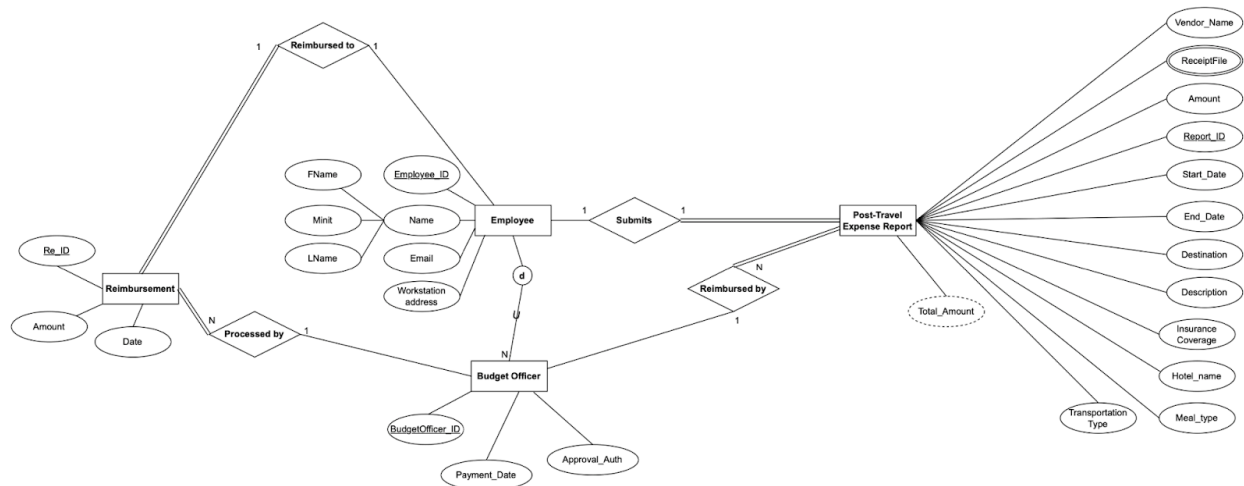
Date: 04/25/2025

Database Functionality Description

The graph-based **travel reimbursement system** we developed in Neo4j is a simplified yet robust representation of our original relational model. The graph database focuses on the most essential entities and their core relationships to support key operations around post-travel expense reimbursement processing.

At the center of the graph is the **Employee** node, which serves as a supernode connecting various functional roles and transaction events. Employees who were traveller submit **Post-Travel Expense Reports**. These reports are linked to a designated **Budget Officer**, who is responsible for processing the reimbursement. Once a report is approved, it is tied to a **Reimbursement** node that stores the disbursed amount and payment date. This reimbursement is further associated with the employee who received the payment and the budget officer who processed it. The graph structure provides a more intuitive view of how data flows, from submission to reimbursement while eliminating the complexity of join tables found in traditional relational schemas.

This transformation allows for a more efficient representation of relationships and makes querying specific paths and interactions between entities easier using Cypher commands. The **corresponding EER diagram** shown below illustrates the simplified model used in Neo4j database model. It highlights the entities included in this phase, their attributes, and the key relationships that were implemented to support graph-based querying and data traversal.



Cypher-Based Data Initialization and Relationship Modeling

We successfully designed a graph database to represent the travel reimbursement system using Neo4j. Instead of relying on traditional tables and rows, we used a graph model that organizes information into two main components:

1. **Nodes:** These represent the key "things" (entities) in our system.
2. **Relationships:** These show how those "things" are connected.

Nodes (Key Entities)

Each node represents an important part of the Travel Reimbursement system. Here's what they are:

1. **Employee:** The Employee entity stores essential information about university personnel who may undertake official travel. This entity helps link each travel request and expense report to the correct individual.

Cypher query

```
CALL {
  LOAD CSV WITH HEADERS FROM 'file:///employee.csv' AS row
  WITH row WHERE row.Employee_ID IS NOT NULL
```

```

MERGE (e:Employee {Employee_ID: toInteger(row.Employee_ID)})
SET e.FName = row.FName,
e.Minit = row.Minit,
e.LName = row.LName,
e.Email = row.Email,
e.Workstation_address = row.Workstation_address
}

```

This query loads employee records from the employee.csv file and merges them into the graph as unique Employee nodes. Each node captures essential personal and contact details of the employee, which serve as the foundation for associating travel and reimbursement activities.

2. **Post-Travel Expense Report:** The Post-Travel Expense Report entity consolidates all actual expenses incurred during the trip. The traveler submits it after returning, attaching receipts and justifications. The Budget Officer then reviews these expenses for accuracy and adherence to policy for final reimbursement approval.

Cypher query

```

CALL {
LOAD CSV WITH HEADERS FROM 'file:///Post_travel_report.csv'
AS row
WITH row WHERE row.Report_ID IS NOT NULL
MERGE (p:post_travel {Report_ID: toInteger(row.Report_ID)})
SET
p.Amount = toFloat(row.Amount),
p.ReceiptFile = toInteger(row.ReceiptFile),
p.Vendor_Name = row.Vendor_Name,
p.Start_Date = datetime(replace(row.Start_Date, ' ', 'T')),
p.End_Date = datetime(replace(row.End_Date, ' ', 'T')),
p.Destination = row.Destination,
p.Description = row.Description,
p.Insurance_Coverage = row.`Insurance Coverage`,
p.Hotel_name = row.Hotel_name,
p.Meal_type = row.Meal_type,

```

```
p.Transportation_Type = row.Transportation_Type
}
```

This command loads post-travel report data from Post_travel_report.csv and creates post_travel nodes representing each expense report. These nodes include various attributes detailing the trip, such as dates, destination, vendor, and total amount, supporting downstream processing and analysis.

3. **Budget Officer:** The Budget Officer entity responsible for allocating funds to employees by processing reimbursements for approved travel expenses. The Budget Officer ensures that disbursements are made promptly and efficiently, facilitating smooth financial transactions for travelers.

Cypher query

```
CALL {
  LOAD CSV WITH HEADERS FROM 'file:///Budget_Officer.csv' AS
  row
  WITH row WHERE row.BudgetOfficer_ID IS NOT NULL
  MERGE (b:budgetofficer {BudgetOfficer_ID:
  toInteger(row.BudgetOfficer_ID)})
  SET
  b.Payment_Date = datetime(replace(row.Payment_Date, ' ',
  'T')),
  b.Approval_Auth = toInteger(row.Approval_Auth)
}
```

This query loads budget officer records from Budget_Officer.csv and creates budget officer nodes. These nodes include attributes such as payment date and approval authority, representing the financial role responsible for approving and processing reimbursements.

4. **Reimbursement:** Facilitate fund allocation to employees after their travel report has been processed and approved. This table connects the payment officer to the reimbursed employee.

Cypher query

```
CALL {  
  LOAD CSV WITH HEADERS FROM 'file:///reimbursement.csv' AS  
  row  
  WITH row WHERE row.Re_ID IS NOT NULL  
  MERGE (r:reimbursement {Re_ID: toInteger(row.Re_ID)})  
  SET  
  r.Amount = toFloat(row.Amount),  
  r.Date = datetime(replace(row.Date, ' ', 'T'))  
}
```

This query loads reimbursement records from reimbursement.csv and creates reimbursement nodes. Each node stores the disbursed amount and the date of reimbursement, representing completed financial transactions.

Relationships (Connections Between Nodes)

We used relationships to connect these nodes, capturing how the entities interact with one another. Here's what they mean:

1. **SUBMITS:** Employees who have traveled submit post-travel expense report detailing their expenses and travel activities.

Cypher query

```
LOAD CSV WITH HEADERS FROM 'file:///Post_travel_report.csv'  
AS pt  
WITH pt WHERE pt.Employee_ID IS NOT NULL  
MATCH (e:Employee {Employee_ID: toInteger(pt.Employee_ID)})  
MATCH (p:post_travel {Report_ID: toInteger(pt.Report_ID)})  
MERGE (e)-[:SUBMITS]->(p);
```

This query creates SUBMITS relationships between Employee nodes and post_travel nodes, indicating which employee submitted each expense report. It models the act of report submission in the system.

- 2. REIMBURSED_BY:** Budget Officers are responsible for reimbursing approved travel expenses based on the submitted post-travel expense reports.

Cypher query

```
LOAD CSV WITH HEADERS FROM 'file:///Post_travel_report.csv'
AS pt
WITH pt WHERE pt.BudgetOfficer_ID IS NOT NULL
MATCH (b:budgetofficer {BudgetOfficer_ID:
toInteger(pt.BudgetOfficer_ID)})
MATCH (p:post_travel {Report_ID: toInteger(pt.Report_ID)})
MERGE (p)-[:REIMBURSED_BY]->(b);
```

This query creates REIMBURSED_BY relationships between post_travel nodes and budgetofficer nodes. It links each report to the officer responsible for reviewing and authorizing reimbursement, allowing traceability of approvals.

- 3. PROCESSED_BY:** All the approved post-travel expense reports were processed by budget officer for reimbursements.

Cypher query

```
LOAD CSV WITH HEADERS FROM 'file:///reimbursement.csv' AS re
WITH re WHERE re.BudgetOfficer_ID IS NOT NULL
MATCH (b:budgetofficer {BudgetOfficer_ID:
toInteger(re.BudgetOfficer_ID)})
MATCH (r:reimbursement {Re_ID: toInteger(re.Re_ID)})
MERGE (r)-[:PROCESSED_BY]->(b);
```

This query defines the PROCESSED_BY relationship between each reimbursement and the budgetofficer who handled its processing. It helps maintain accountability in the reimbursement workflow.

4. **REIMBURSED_TO:** Employees got reimbursed after their travel report had been processed and approved. This relationship connects the reimbursement entity back to the reimbursed employee.

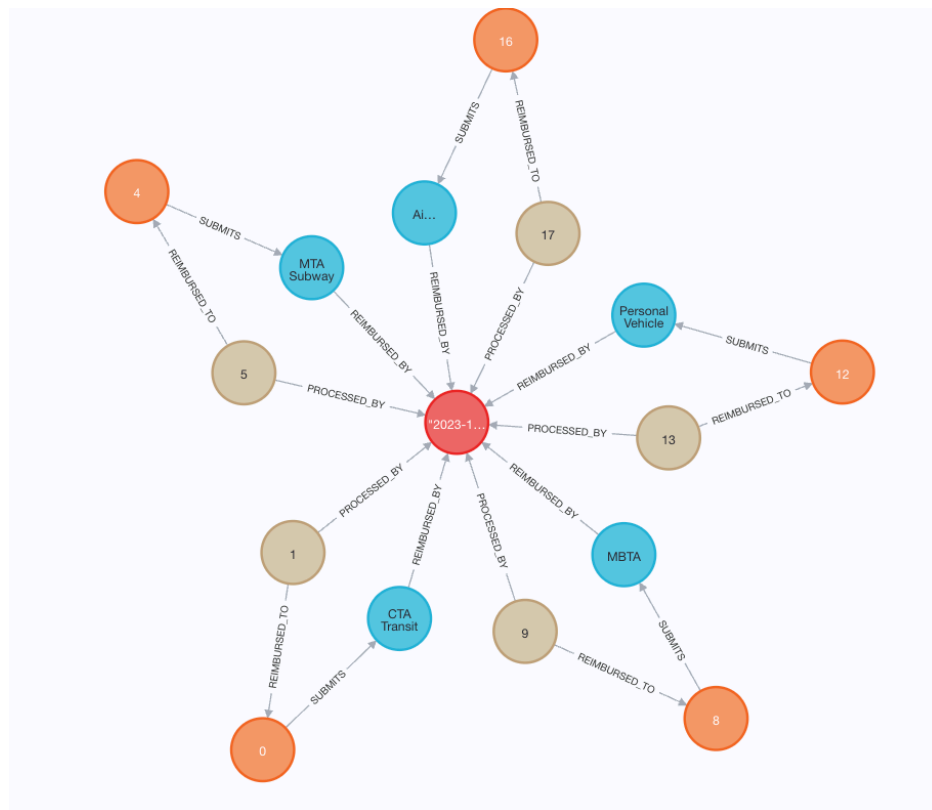
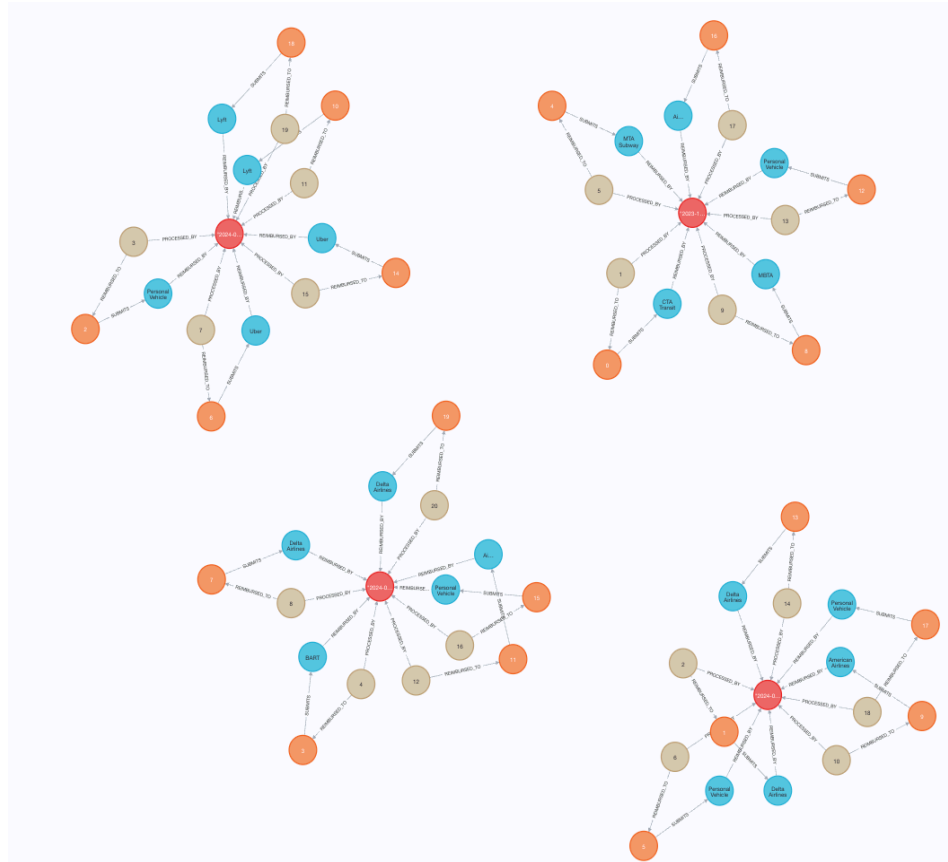
Cypher query

```
LOAD CSV WITH HEADERS FROM 'file:///reimbursement.csv' AS re
WITH re WHERE re.Employee_ID IS NOT NULL
MATCH (e:Employee {Employee_ID: toInteger(re.Employee_ID)})
MATCH (r:reimbursement {Re_ID: toInteger(re.Re_ID)})
MERGE (r)-[:REIMBURSED_TO]->(e);
```

This query builds the REIMBURSED_TO relationship from reimbursement nodes to the corresponding Employee nodes who received the funds. It completes the cycle of the travel reimbursement process by linking disbursed payments to recipients.

Graph Database

The diagram of the graphical database is shown below.



Relationships and Nodes Deletion

Deleting Relationships

These queries target specific connections (relationships) between different types of data points (nodes) in your graph.

Delete SUBMITS relationships:

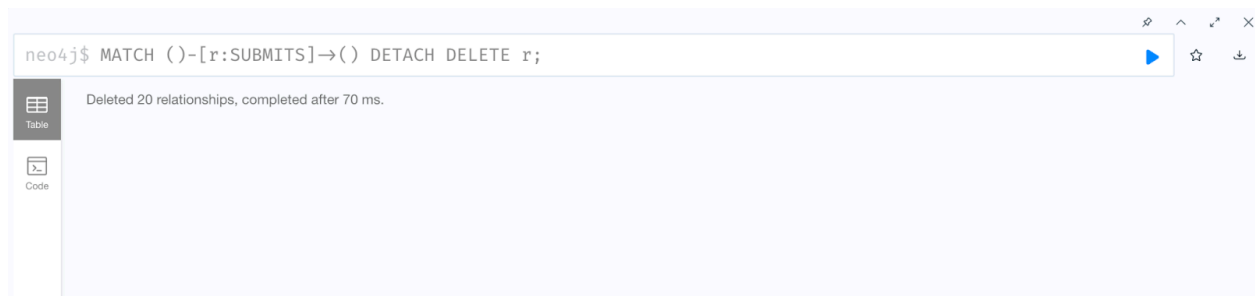
Cypher query

```
MATCH ()-[r:SUBMITS]->()
DETACH DELETE r;
```

Purpose: To remove the connections indicating that an Employee submitted a Post-Travel Expense Report.

Details:

1. `MATCH ()-[r:SUBMITS]->()`: This finds all relationships (r) in the database that have the type SUBMITS. The empty parentheses () signify that it matches the relationship regardless of the specific nodes it connects.
2. `DETACH DELETE r`: This deletes the matched relationships (r). DETACH isn't strictly necessary for deleting relationships themselves but is included for consistency and doesn't harm.



Deleting Nodes

These queries target the actual data points (nodes) themselves. Using DETACH DELETE is crucial here, as it ensures any relationships still connected to these nodes are removed along with the node, preventing errors.

Delete Employee nodes:

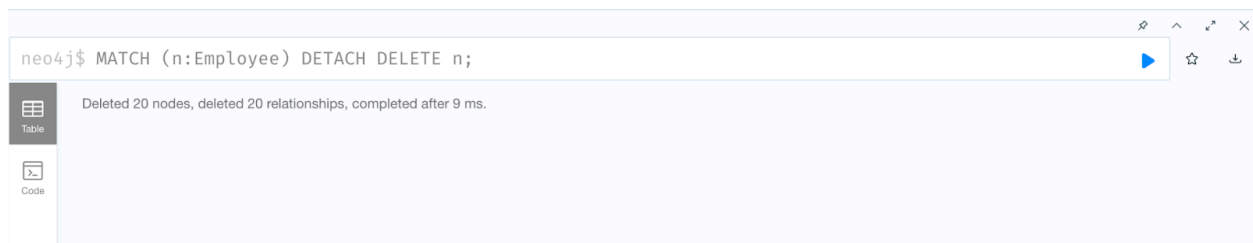
Cypher query

```
MATCH (n:Employee)
DETACH DELETE n;
```

Purpose: To remove all employee records from the database.

Details:

1. MATCH (n:Employee): Finds all nodes (n) that have the label Employee.
2. DETACH DELETE n: Deletes the matched Employee nodes (n) and any relationships connected to them (e.g., SUBMITS, REIMBURSED_TO relationships if not already deleted).



Data Retrieval Questions and Explanations:

1. Find all travel expense reports submitted by a specific employee (e.g., Employee ID 1).



Rationale: This question helps track the submission history for a particular employee. It uses the SUBMITS relationship.

Cypher Query

```
MATCH (e:Employee {Employee_ID: 1})-[s:SUBMITS]->(p:post_travel)
RETURN e.FName, e.LName, p.Report_ID, p.Destination, p.Amount,
p.Start_Date, p.End_Date
```

Explanation: This query finds the Employee node with Employee_ID 1, follows the outgoing SUBMITS relationship to find connected post_travel report nodes, and returns the employee's name along with details of their submitted reports.

Output:

	e.FName	e.LName	p.Report_ID	p.Destination	p.Amount	p.Start_Date	p.End_Date
1	"John"	"Doe"	1	"Chicago"	1200.5	"2024-01-01T10:00:00Z" 	"2024-01-05T18:00:00Z" 

2. Which budget officer reimbursed a specific travel report (e.g., Report ID 5)?

Rationale: This helps identify who approved or processed the reimbursement for a specific travel claim. It uses the REIMBURSED_BY relationship.

Cypher Query

```
MATCH (p:post_travel {Report_ID:
5})-[rb:REIMBURSED_BY]->(b:budgetofficer)
RETURN p.Report_ID, p.Destination, b.BudgetOfficer_ID
```

Explanation: This query locates the post_travel report with Report_ID 5, follows the REIMBURSED_BY relationship to the budgetofficer node, and returns the report details along with the ID of the budget officer who reimbursed it.

Output:

	p.Report_ID	p.Destination	b.BudgetOfficer_ID
1	5	"New York"	5

3. Find all employees who received a reimbursement processed by a specific budget officer (e.g., Budget Officer ID 10).

Rationale: This question identifies all employees whose reimbursements were handled by a particular budget officer. It uses the PROCESSED_BY and REIMBURSED_TO relationships.

Cypher Query

```
MATCH (b:budgetofficer {BudgetOfficer_ID:
10})<-[pr:PROCESSED_BY]-(r:reimbursement)-[rt:REIMBURSED_TO]->(e:
Employee)
RETURN b.BudgetOfficer_ID, r.Re_ID, r.Amount, e.Employee_ID,
e.FName, e.LName
```

Explanation: This query starts from the budgetofficer with ID 10, finds associated reimbursement nodes via the incoming PROCESSED_BY relationship, then finds the Employee nodes connected to those reimbursements via the REIMBURSED_TO relationship, returning details about the officer, reimbursement, and employee.

Output:

b.BudgetOfficer_ID	r.Re_ID	r.Amount	e.Employee_ID	e.FName	e.LName
10	2	890.0	2	"Jane"	"Smith"
10	6	990.9	6	"Dan"	"Miller"
10	10	1150.25	10	"Hank"	"Anderson"
10	14	1380.4	14	"Leo"	"Harris"
10	18	860.9	18	"Paul"	"Lee"

4. List all travel reports with an amount greater than \$1200.

Rationale: This helps identify high-value expense reports for review or analysis. It filters nodes based on a property value.

Cypher Query

```
MATCH (p:post_travel)
WHERE p.Amount > 1200
RETURN p.Report_ID, p.Destination, p.Amount, p.Description
```

Explanation: This query finds all `post_travel` nodes, filters them based on the `Amount` property being greater than 1200, and returns details of those specific reports.

Output:

	p.Report_ID	p.Destination	p.Amount	p.Description
1	1	"Chicago"	1200.5	"Tech conference"
2	3	"Denver"	1350.75	"Annual sales meeting"
3	5	"New York"	1450.0	"Attending industry conference"
4	7	"Miami"	1235.5	"Conducting field research"
5	14	"Dallas"	1380.4	"Skill-building workshop"
6	19	"Las Vegas"	1240.6	"Skill-building workshop"

5. How many reimbursement claims has each budget officer processed?

Rationale: This provides an overview of the workload distribution among budget officers. It uses the `PROCESSED_BY` relationship and the `COUNT` aggregation function.

Cypher Query

```

MATCH (b:budgetofficer)<-[pr:PROCESSED_BY]-(r:reimbursement)
RETURN b.BudgetOfficer_ID, COUNT(r) AS
NumberOfReimbursementsProcessed
ORDER BY NumberOfReimbursementsProcessed DESC

```

Explanation: This query finds all budget officer nodes, counts the number of incoming PROCESSED_BY relationships from reimbursement nodes for each officer, and returns the officer's ID and the total count, ordered from highest to lowest.

Output:

	b.BudgetOfficer_ID	NumberOfReimbursementsProcessed
1	5	5
2	10	5
3	15	5
4	20	5

Project Challenges and Resolution Strategies

Transitioning from a relational database structure to a graph-based model in Neo4j presented several practical and conceptual challenges. One of the earliest hurdles we faced was ensuring compatibility between our CSV files and Neo4j's expected formats. **Although our CSVs were derived from the relational schema built in MySQL, many of the fields, especially date and time values had to be manually restructured to comply with Neo4j's datetime() parsing rules.** Initially, Neo4j returned parsing errors for standard SQL datetime formats (e.g., 2024-01-01 10:00:00), which required us to either convert them to ISO format (2024-01-01T10:00:00) or store them as strings to avoid disruption.

Another challenge we faced was of **determining how much of our original schema to simplify**. In the interest of clarity and assignment scope, we decided to limit our Neo4j model to a focused set of core entities and relationships that best represent the reimbursement workflow from employee submission to report review and final payment.

These experiences highlighted the importance of thoughtful data preparation, close attention to Neo4j's syntax expectations, and iterative testing. By the end of this phase, we not only overcame these technical obstacles but also gained a deeper understanding of how graph databases can simplify complex data interactions in ways that relational models cannot easily replicate.

Conclusion

This Neo4j graph database offers a simplified yet powerful representation of the travel reimbursement process, capturing the essential relationships between employees, post-travel expense reports, and budget officers. By modeling key interactions as nodes and relationships, the system supports intuitive and efficient data traversal, particularly for tracking report submissions, approvals, and reimbursements. The use of Cypher queries enables clear insights into employee activity, financial workflows, and overall reimbursement trends, making the graph model both functional for current needs and scalable for future enhancements.