## Project Planning

Introduction

The Gift of the Givers Foundation web application is made to effectively oversee all aspects of the organization's operations, such as resource allocation, volunteer coordination, gift processing, and user management. In terms of functionality, the system enables users to sign up, get in, and maintain their accounts. Users can have different degrees of access depending on what role they play, such as administrators, volunteers, and donations. While funders can make safe online contributions and keep track of their payments, volunteers can register for relief programs and receive updates about their activities. In addition, the system makes project administration easier by enabling administrators to plan and oversee relief initiatives, guaranteeing that funds are used effectively. In order to serve a varied user base, the program also offers multi-language support, email notifications, and in-app messaging for user communication.

Functional Requirements

**User Account Creation and Access**

* It must be possible for users to register using their name, email address, and password.
* Secure authentication should be supported by the system for login functionality.

Donor Administration

* Online contributors ought to be allowed to send money via safe payment gateways.
* Donor receipts should be generated by the system and donations should be tracked

**Coordination of Volunteers**

* Volunteers should be able to sign up for specific relief projects and examine their allocated tasks.
* The system should allow for scheduling volunteer activities and sending reminders to volunteers.

Notifications and Communication

* Emails and in-app messaging between users should be made easier by the application.
* Users should receive notifications when there are significant updates, including new initiatives, volunteer opportunities, or acknowledgments of donations.

Non-Functional Requirements

**Accessible**

* To guarantee that the application is still accessible during maintenance or unplanned downtime, redundancy should be put into place.

**Reliability**

* Modularity should be considered in the system's architecture to facilitate simple updates and maintenance.
* It is necessary to have clear documentation in order to facilitate future development and continued maintenance.

**Data Accuracy**

* With safeguards against data loss or corruption, the system must guarantee that all data is accurate and consistent.
* In order to maintain data integrity during processes like resource allocation and donations, the application needs include transaction management.

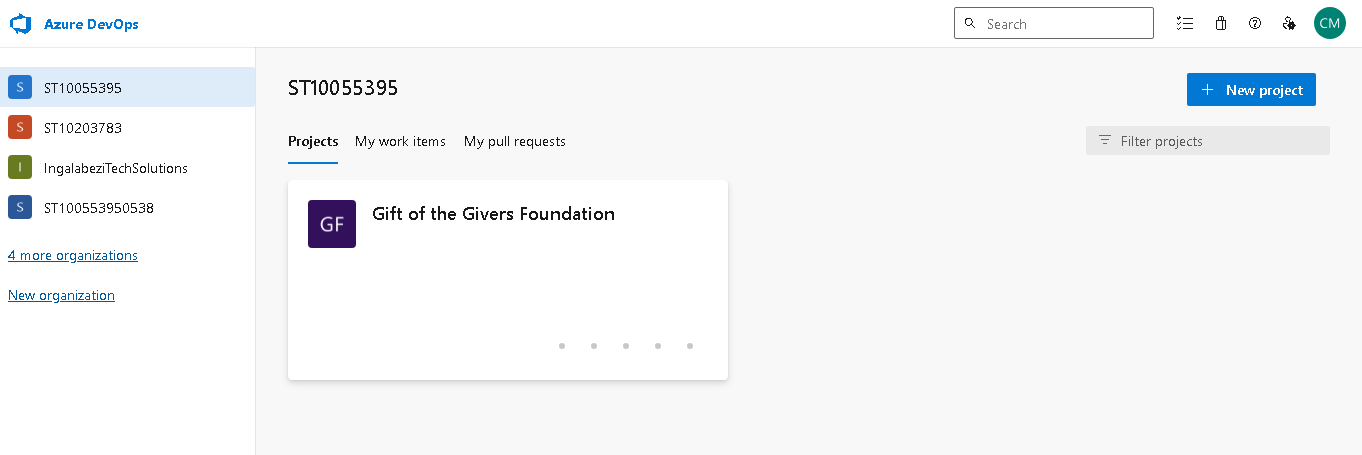
**Restore and Backup**

* Regular data backups should be performed by the system, and in the event of a breakdown, a recovery plan should be in place to restore data.
* The user experience and system performance should be as unaffected as possible by the backup process.

# 1.1Setting up Azure Boards Project

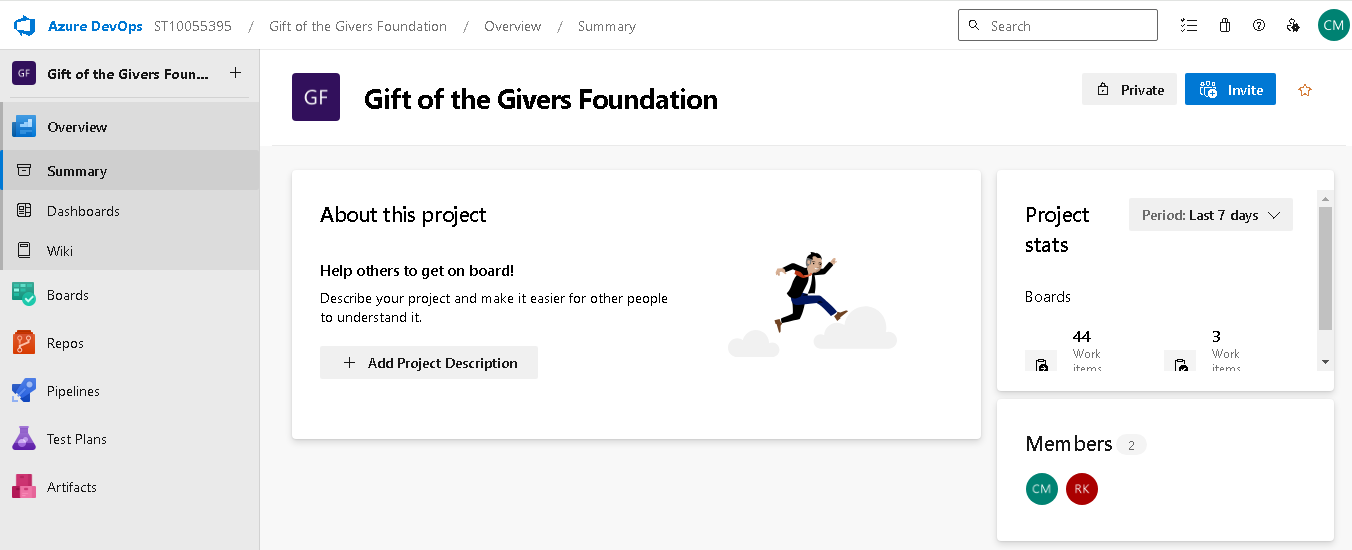
Step 1

* Under the name Gift of the Givers Foundation, I have developed an Azure Boards project.

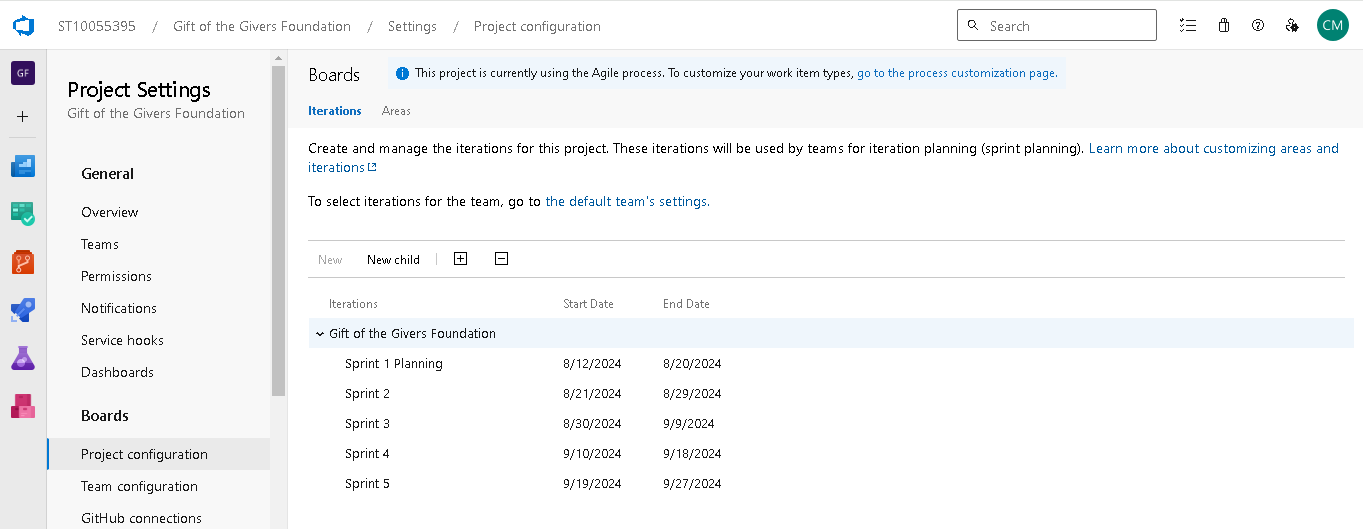


Step 2

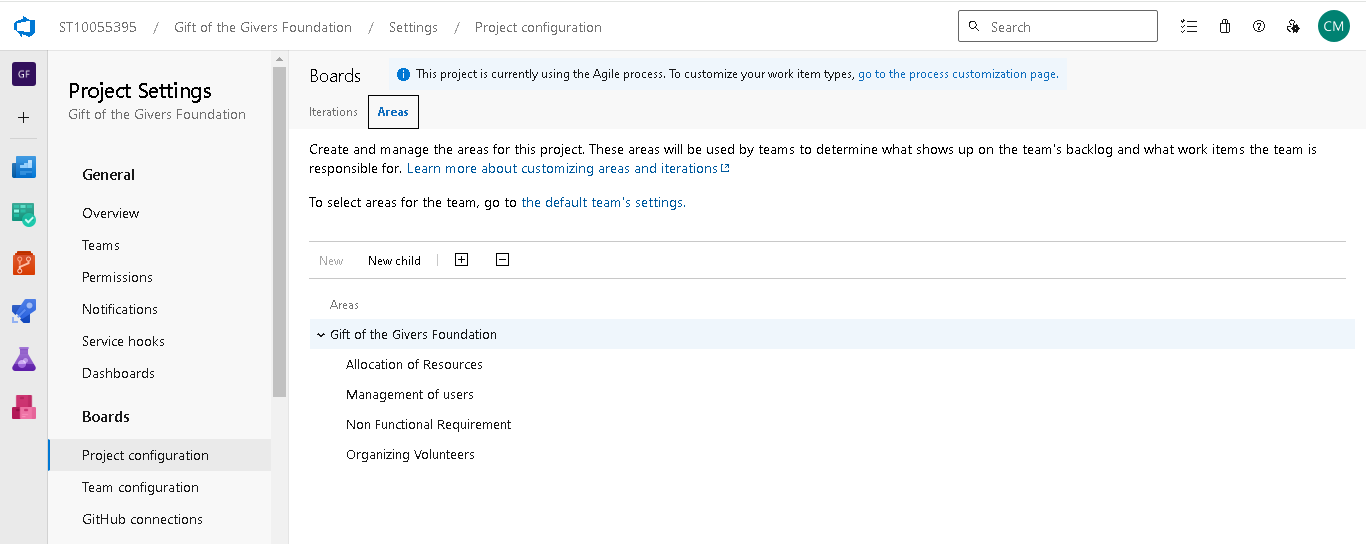
* Celebrate Mashaba and Rhulani Kobeni are the two members of the project that I assigned.



* This are iteration of the project



* And the Area of the project



# 1.2 Creating User Stories, Epics, and Tasks

Step 3

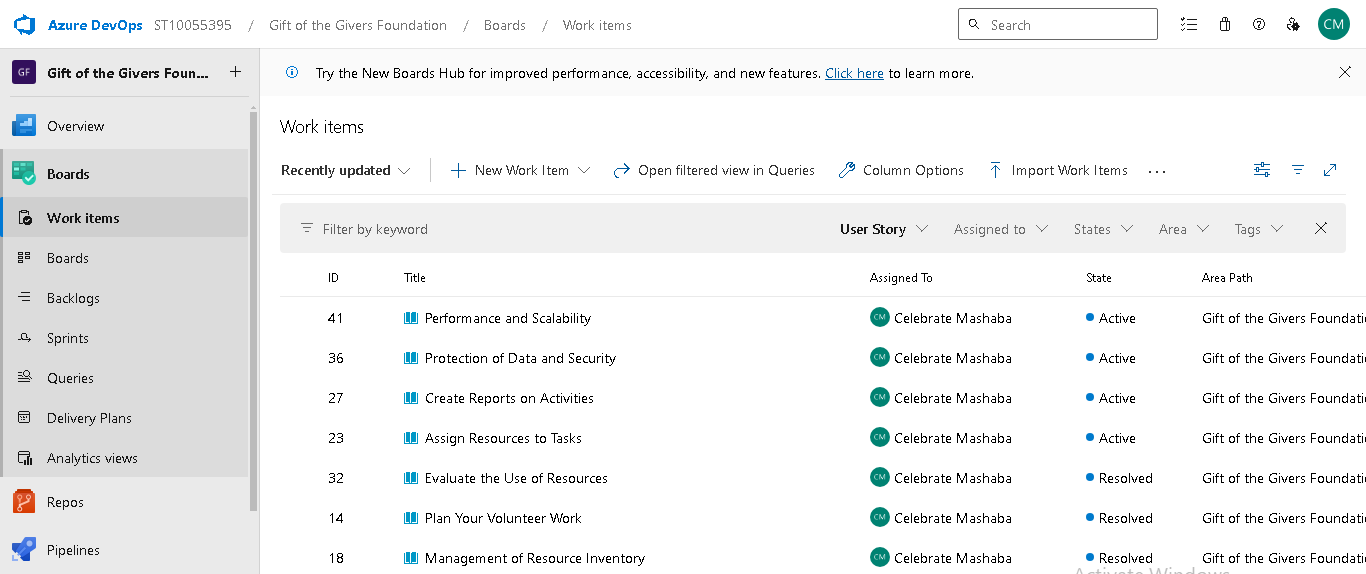
Allocation of task, epic, user stories

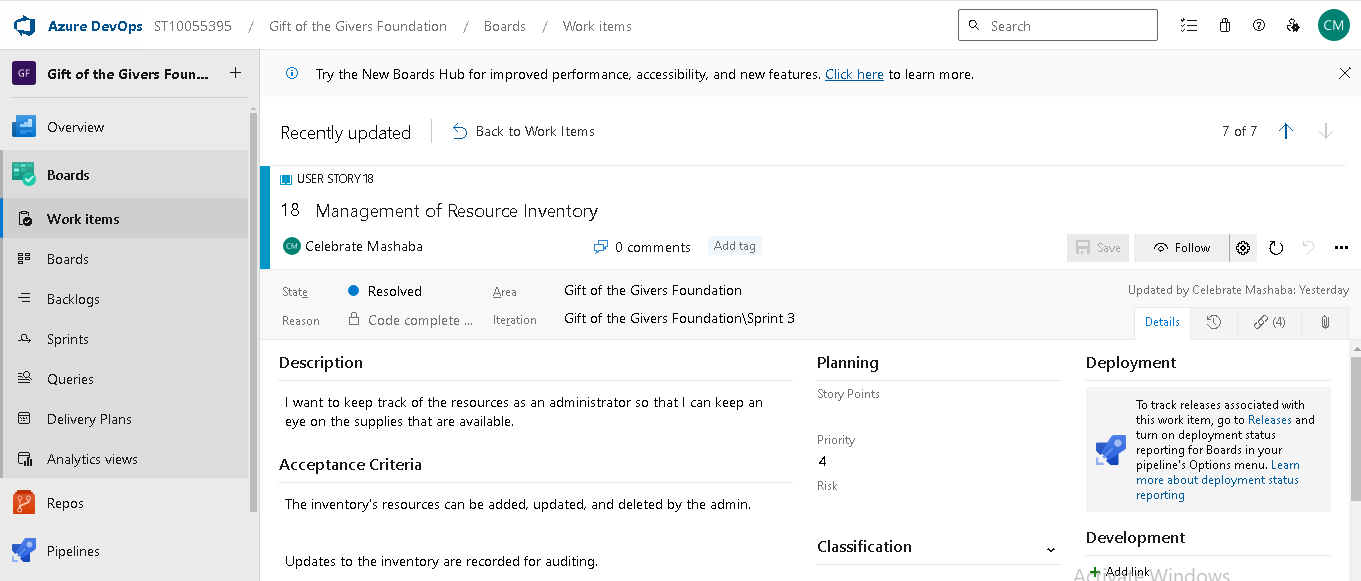
A screenshot of a computer

Description automatically generated

In this phase, I have established a user story, and each user story contains a task user story, an epic parent, a description, and accepting criteria.

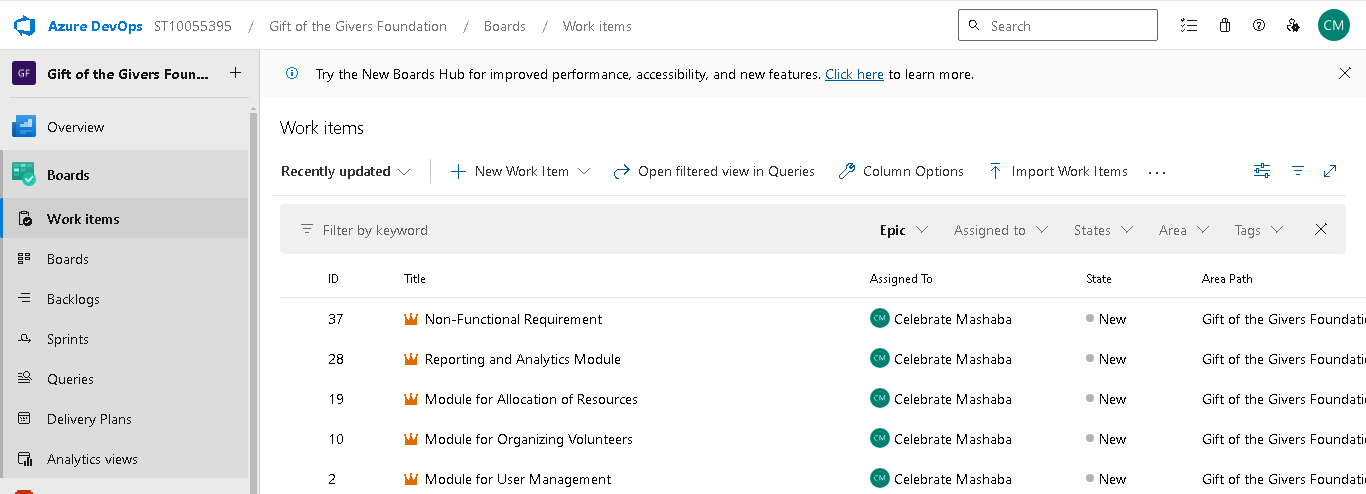
**User Story**

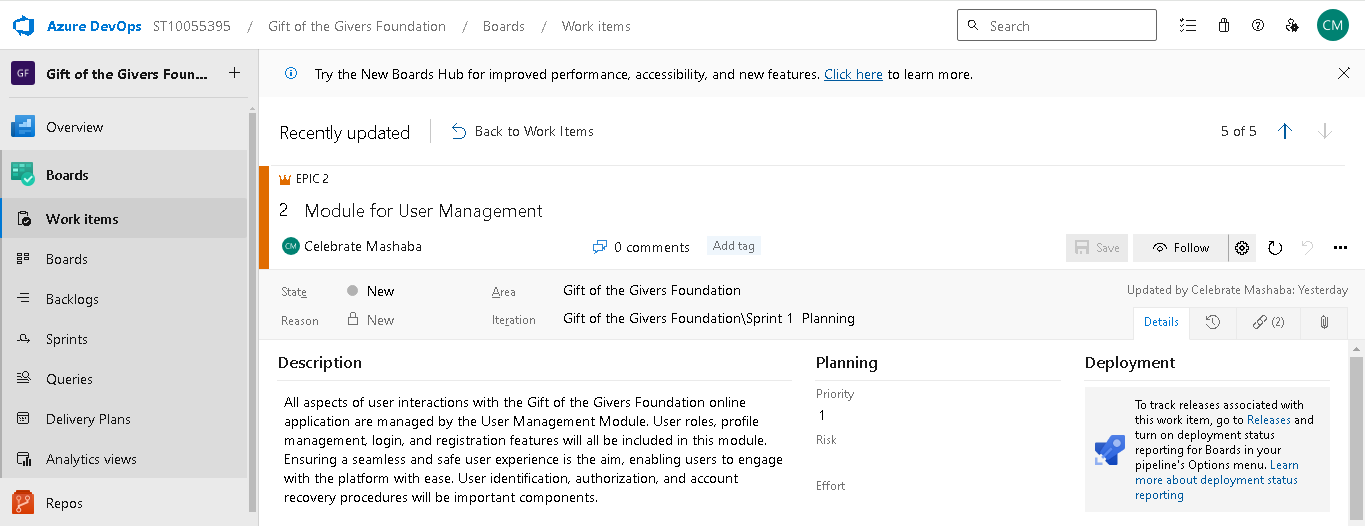




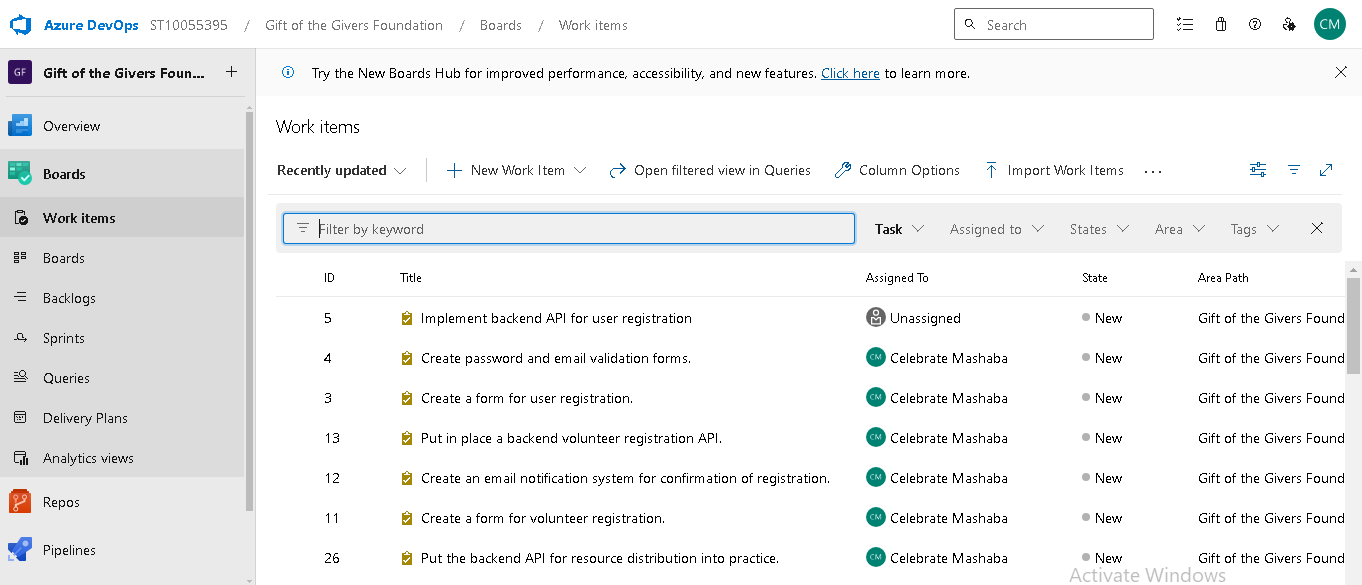
• After that, epics were made, which are the parent stories of user stories and also have tasks and descriptions.

**Epics**





And this are tasks and each task has its own parent and user story



Step 4

This is my board, where my work items and user stories are shown. The items are divided into new, active, resolved, and closed categories, and each task and user story has been assigned to a specific category.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

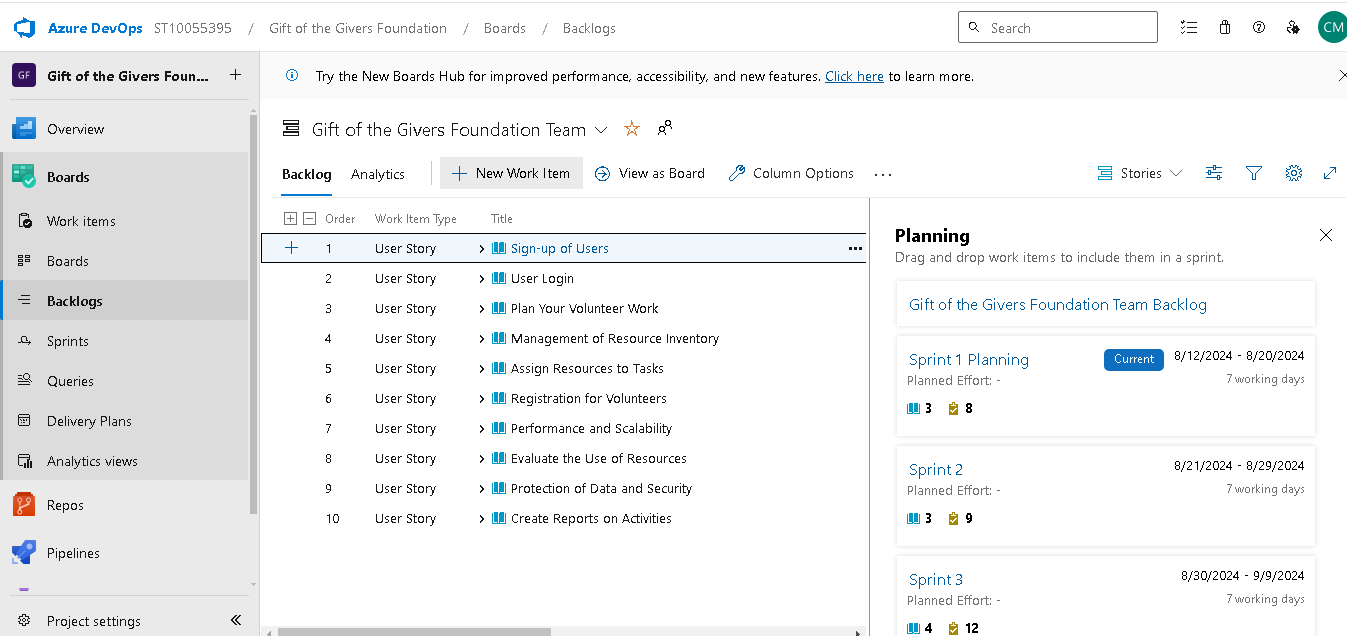
Description automatically generated

A screenshot of a computer

Description automatically generated

This is my backlog, which shows you the features and user stories I have for the web application as well as the number of tasks and user stories assigned to each sprint so you can track the development of the program.

Backlog

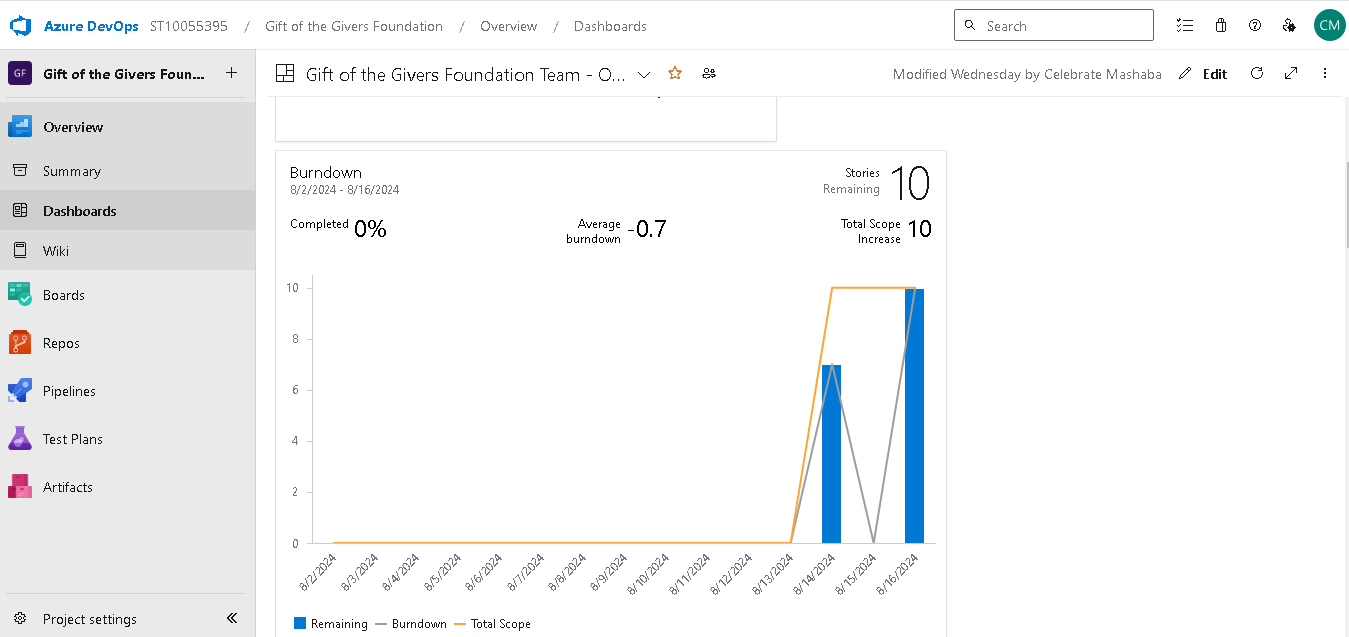


This is my dashboard. I can see my progress by using the burndown structure to display my work items.

Dashboard

A screenshot of a computer

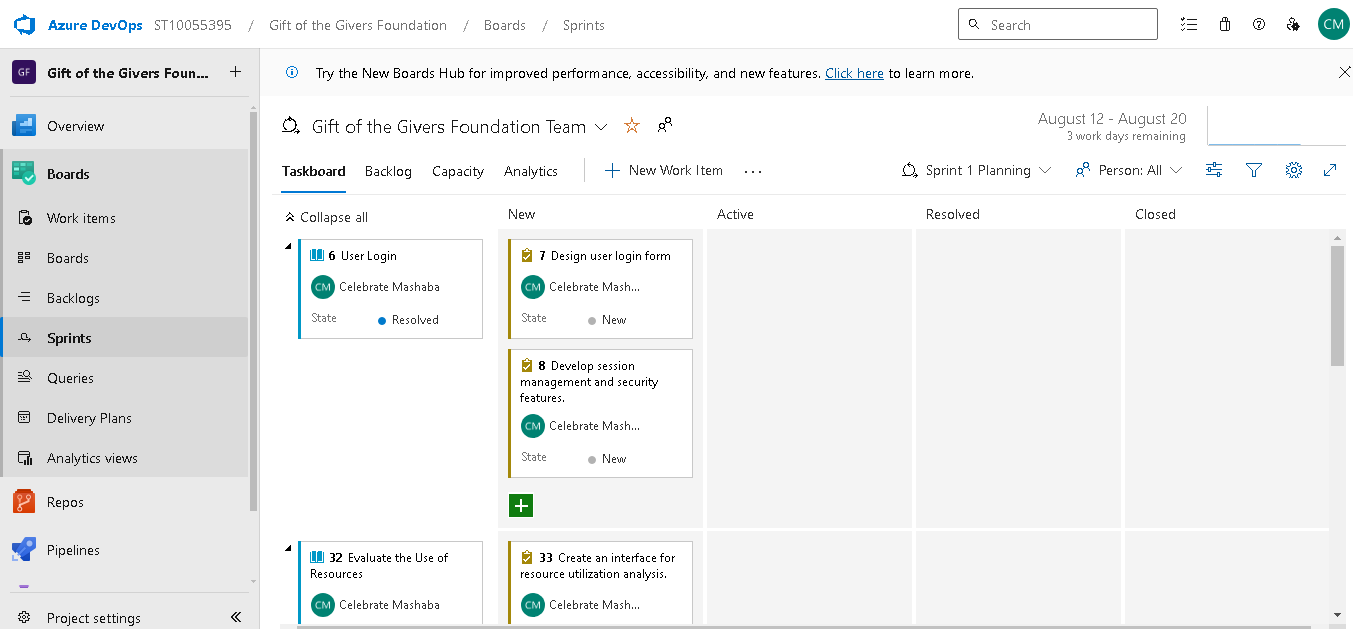
Description automatically generated

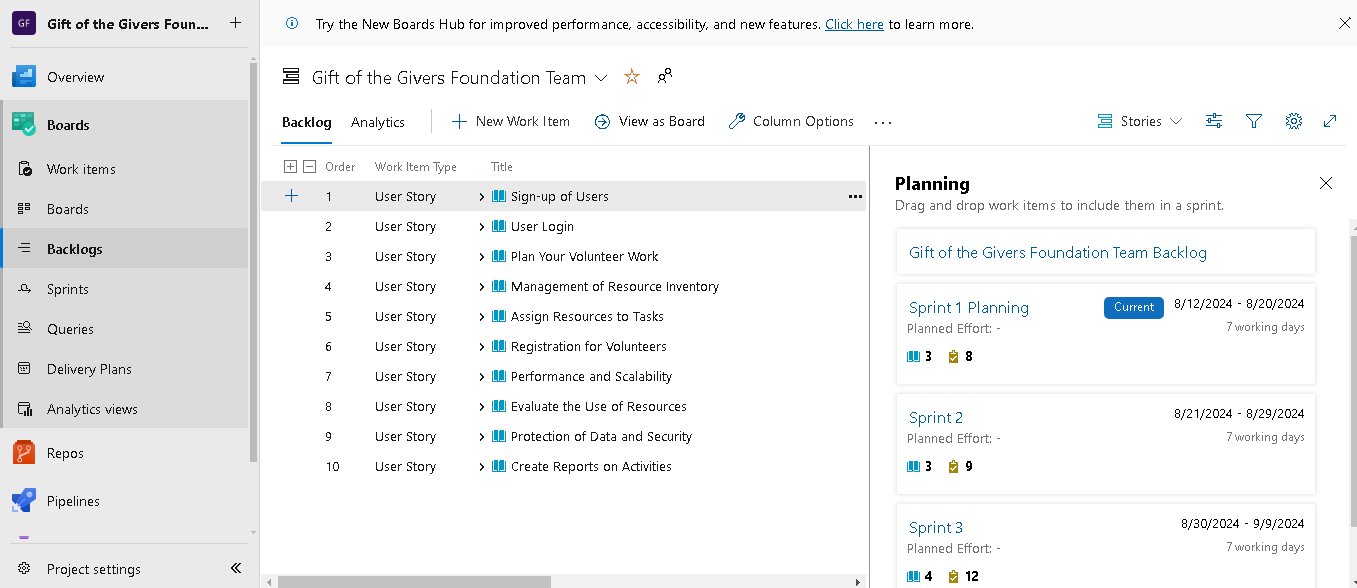


Step 5

* 1. Organize Work item into Sprint

I divided the project timeline into iterative development which are sprint and I assigned user stories and task to each sprint





2.1 Understanding Database Design Principles:

**Importance of Database Design in Ensuring Efficiency, Scalability, and Data Integrity**

**Data integrity:**

* Data normalization: normalization ensures data consistency, reduces data anomalies, and improves data quality
* Constraints and triggers: constraints and triggers enforces data integrity rules, preventing invalid data entry
* Backup and recovery: regular backups and recovery plan ensures data availability and minimize data loss

**Scalability**

* Managing Growth: When user counts, data volumes, or transaction volumes rise, a scalable database architecture can support greater loads without compromising performance.
* Distributed Systems: In order to manage larger datasets and higher traffic levels, scalability also includes the capacity to distribute the database across several servers or locations.
* Flexible schema design: a flexible schema accommodation changing data structures and growing data volumes

**Efficiency:**

* Query Performance: A well-designed database ensures that information is retrieved quickly and efficiently, saving time and resources.
* Storage Optimization: Compact and logical data storage, reduced redundancy, and optimal storage use are all guarantees of a well-designed database
* Improved data organization: logical data organization enables easy access and manipulation data

**Additional benefit of good database design include:**

* Improved data security
* Enhanced data sharing and collaboration
* Better decision-making through data analysis
* Reduces maintenance and support costs
* Improved scalability and adaptability to changing requirement

Key Database Design Principle:

**Normalization**

Normalization: Normalization is the process of organizing data to lessen duplication and improve data integrity.

* First normal form (1NF): Each table cell contains a single value
* Second normal form (2NF): Each non-key attribute depends on the entire primary key
* Third normal form (3NF): If a table is in 2NF, and a non-key attribute, then it should be moved to a separate table  
    
  Tables: Users, Roles, and UserRoles in the User Management Module   
  Normativeization Sort user data into three categories: Roles (role definitions), UserRoles (role affiliations), and Users (generic user information).   
  Benefits: Reduces unnecessary data and ensures consistent role allocations.

**Indexing**

Definition of indexing: Indexing is the process of building a data structure to facilitate faster database access for table data.

* Primary index: A unique on the primary key column(s)
* Secondary index: An index on a non-primary key column
* Composite index: An index on multiple columns
* Unique index: ensures uniqueness of values in a column  
    
  For instance:   
  Indexing   
  Tables: Volunteer Tasks and the Volunteer Coordination Module   
  Indexing: Create indexes for frequently searched data, including TaskID and VolunteerID.   
  Benefits: It speeds up search and retrieval operations, particularly when connecting volunteers to jobs based on preset criteria.

**Referential integrity**

Referential Integrity: Definition: Referential integrity ensures consistent relationships between tables. It is enforced by using foreign keys that reference primary keys in related tables.

* Primary key: A unique identifier for each row in a table
* Foreign key: A field in a table that links to the primary key of another table
* Referential integrity constraints: Ensures data consistency between related table

Example:  
Tables in the Resource Allocation Module: Resources, ResourceAllocations   
Ensure Foreign Key Resource Allocations.Materials with ResourceID citations.ResourceID.   
Cascading Actions: To remove allocations in the case that a resource is destroyed, utilize cascading delete.   
Benefits: Maintains consistency and ensures current and valid resource allocations

# 2.2Designing the Database Schema:

Identify the entities and attributes relevant to the Gift of the Givers Foundation web application (e.g., users, donations, relief projects).

Entity : My\_**User**

* Attributes: U\_ID (Primary Key), FirstName, LastName, Email, Password, R\_ID (Foreign Key)

Entity : **Volunteers**

* Attributes: V\_ID (Primary Key), U\_ID (Foreign Key), Availability, Skills

Entity :My\_**Roles**

* Attributes: R\_ID (Primary Key), RoleName, Description

Entity : **Donations**

* Attributes: D\_ID (Primary Key), U\_ID (Foreign Key), Amount, DonationDate, PaymentMethod

Entity : **R\_Projects**

* Attributes: P\_ID (Primary Key), ProjectName, Description, StartDate, EndDate, Status

Entity : **Volunteer\_T**

* Attributes: T\_ID (Primary Key), P\_ID (Foreign Key), V\_ID (Foreign Key), TaskName, TaskDescription, StartDate, EndDate, Status

Entity : **Resources**

* Attributes: R\_ID (Primary Key), ResourceName, Quantity, Description

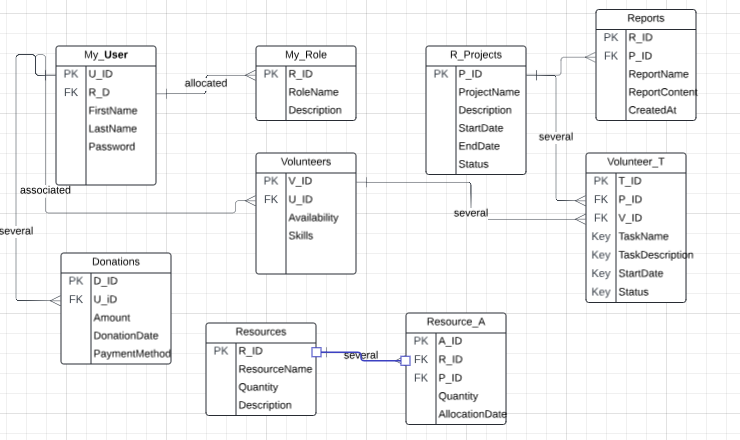
Entity : **Resource\_A**

* Attributes: A\_ID (Primary Key), R\_ID (Foreign Key), P\_ID (Foreign Key), Quantity, AllocationDate

Entity : **Reports**

* Attributes: R\_ID (Primary Key), P\_ID (Foreign Key), ReportName, ReportContent, CreatedAt

### Entity-Relationship Diagram (ERD)



1. **My\_Users and My\_Roles**

Relationship: One-to-Many (one role can be allocated to many users, but a user has only one role).**Foreign Key:My\_** Users.R\_ID references My\_Roles.R\_ID.

1. **My\_Users and Volunteers**

Relationship: One-to-One (each volunteer is associated with one user, and vice versa).

**Foreign Key:** Volunteers.U\_ID references My\_Users.U\_ID.

1. **My\_Users and Donations**

Relationship: One-to-Many (one user can make many donations, but a donation is made by one user).**Foreign Key:** Donations.U\_ID references My\_Users.U\_ID.

1. **R\_Projects and Volunteer\_T**

Relationship: One-to-Many (one project can have many tasks, but a task belongs to one project).

**Foreign Key:** Volunteer\_T.P\_ID references R\_Projects.P\_ID.

1. **Volunteers and Volunteer\_T**

Relationship: One-to-Many (one volunteer can have many tasks, but a task is assigned to one volunteer).

**Foreign Key:** Volunteer\_T.V\_ID references Volunteers.V\_ID.

1. **R\_Projects and ResourceAllocations**

Relationship: One-to-Many (one project can have many resource allocations, but an allocation is for one project).

**Foreign Key:** Resource\_A.P\_ID references R\_Projects.P\_ID.

1. **Resources and ResourceAllocations**

Relationship: One-to-Many (one resource can have many allocations, but an allocation involves one resource).

**Foreign Key:** Resource\_A.R\_ID references Resources.R\_ID.

1. **R\_Projects and Reports**
2. Relationship: One-to-Many (one project can have many reports, but a report is for one project).

**Foreign Key:** Reports.P\_ID references ReliefProjects.P\_ID.

**Entity-Relationship Diagram (ERD)**

### Defining Tables, Columns, and Data Types

CREATE TABLE My\_Users (

U\_ID INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(100),

LastName NVARCHAR(100),

Email NVARCHAR(255) UNIQUE,

Password NVARCHAR(255),

R\_ID INT

FOREIGN KEY (R\_ID) REFERENCES Roles(R\_ID)

CREATE TABLE Volunteers (

V\_ID INT PRIMARY KEY IDENTITY,

U\_ID INT,

Availability NVARCHAR(255),

Skills NVARCHAR(255),

FOREIGN KEY (U\_ID) REFERENCES Users(U\_ID)

);

**Roles Table**

CREATE TABLE MY\_Roles (

R\_ID INT PRIMARY KEY IDENTITY,

RoleName NVARCHAR(100),

Description NVARCHAR(255)

);

**Donations Table**

CREATE TABLE Donations (

D\_ID INT PRIMARY KEY IDENTITY,

U\_ID INT,

Amount DECIMAL(10, 2),

DonationDate DATETIME,

PaymentMethod NVARCHAR(50),

FOREIGN KEY (U\_ID) REFERENCES Users(U\_ID)

);

**R\_Projects Table**

CREATE TABLE R\_Projects (

P\_ID INT PRIMARY KEY IDENTITY,

ProjectName NVARCHAR(255),

Description NVARCHAR(1000),

StartDate DATETIME,

EndDate DATETIME,

Status NVARCHAR(50)

);

**VolunteerTasks Table**

CREATE TABLE Volunteer\_T(

T\_ID INT PRIMARY KEY IDENTITY,

P\_D INT,

V\_ID INT,

TaskName NVARCHAR(255),

TaskDescription NVARCHAR(1000),

StartDate DATETIME,

EndDate DATETIME,

Status NVARCHAR(50),

FOREIGN KEY (P\_ID) REFERENCES R\_Projects(P\_ID),

FOREIGN KEY (V\_ID) REFERENCES Volunteers(V\_ID)

);

**Resources Table**

CREATE TABLE Resources (

R\_ID INT PRIMARY KEY IDENTITY,

ResourceName NVARCHAR(255),

Quantity INT,

Description NVARCHAR(1000)

);

**Resource\_A Table**

CREATE TABLE Resource\_A (

A\_ID INT PRIMARY KEY IDENTITY,

R\_ID INT,

P\_ID INT,

Quantity INT,

AllocationDate DATETIME,

FOREIGN KEY (R\_ID) REFERENCES Resources(R\_ID),

FOREIGN KEY (P\_ID) REFERENCES R\_Projects(P\_ID)

);

**Reports Table**

CREATE TABLE Reports (

R\_D INT PRIMARY KEY IDENTITY,

P\_ID INT,

ReportName NVARCHAR(255),

ReportContent NVARCHAR(MAX),

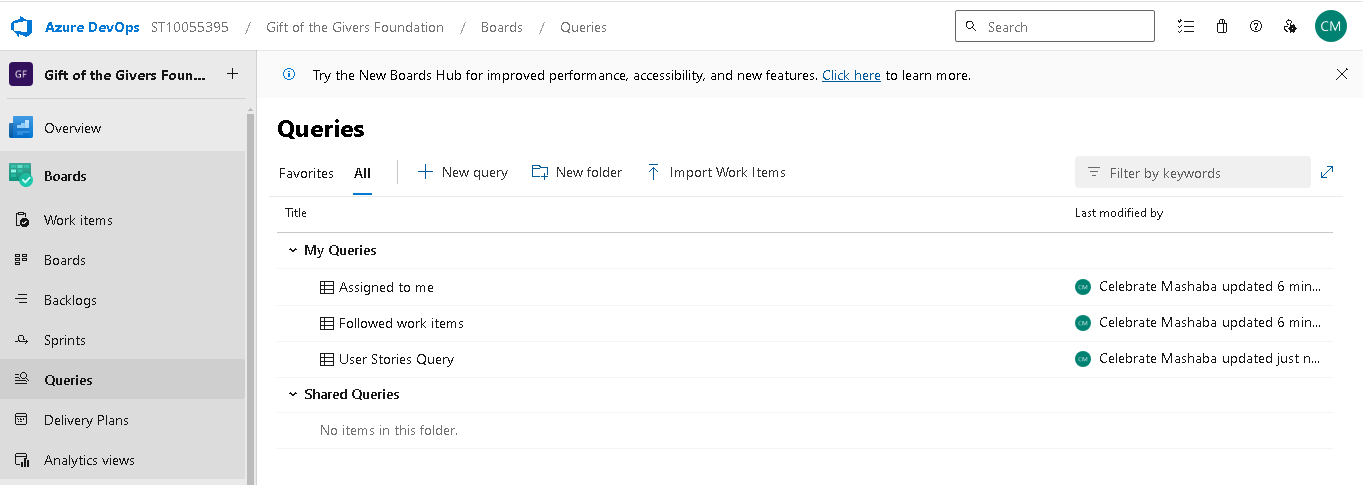
CreatedAt DATETIME DEFAULT GETDATE(),

FOREIGN KEY (P\_ID) REFERENCES R\_Projects(P\_ID)

);

# 2.3 Optimising database Performance

Queries

 • Indexing Techniques   
One of the most effective strategies for enhancing query performance is an index. They facilitate fast data retrieval from the database without requiring a full table scan.   
Apply a clustered index on the primary key columns. By establishing the physical order of the data in the table, this index enables faster data retrieval range queries.   
Non-Clustered Index: On columns that are often used in the WHERE, JOIN, ORDER BY, and GROUP BY clauses, create non-clustered indexes. For example, indexing the DonationDate column can greatly enhance efficiency if queries are frequently used to filter donations by date.

Covering Indexes: Create indexes that contain every column that a query uses. By doing this, the database may completely satisfy the query using the index and avoid doing further lookups in the underlying table.   
Indexing using Selection: Refrain from over-indexing since it can cause write operations (INSERT, UPDATE, and DELETE) to lag. Pay particular attention to indexing columns that are often requested or that facilitate effective table joining.   
• Query Enhancement   
Improving your SQL queries is crucial to reducing database load and accelerating response times.

Employ Execution Plans: Examine your queries' execution plans to find slow-running or full-table scan-causing queries. Based on the results, modify the searches by adding or removing indexes.   
Steer clear of SELECT \*: Selecting more than the necessary columns is always preferable to using SELECT \*, which might result in more I/O and needless data retrieval.   
Set parameters for your queries: To guarantee that SQL Server may reuse execution plans and minimize the expense of query parsing and compilation, utilize parameterized queries.   
Batch Queries: To cut down on database round-trips, bundle several queries into one wherever feasible.

• Partitioning  
By splitting huge tables into smaller, easier-to-manage sections, partitioning makes table management easier. This can reduce the quantity of data the database needs to scan, which can enhance performance.   
Partition huge tables horizontally by range; for example, divide the Donations table according to date. As a result, queries can now scan only the pertinent partitions rather than the full table.   
Dividing a table vertically can be done if certain columns are not frequently accessed. To make the primary table smaller, for example, user-related data that is not commonly utilized can be moved to a separate table.