Group 7

Freelancer Collaboration Platform

Secure, Smart, and Scalable

Team Members

Tanvi Bollavaram

Bhuvaneshwari Kalvakuntla

Fang Yuan

Sumanth Anumula

Neha Palanati

Project Overview

Objective

To build a secure, scalable, and relational database that powers a full-featured freelancer collaboration platform—supporting smart team matching, project management, and milestone-based payments.

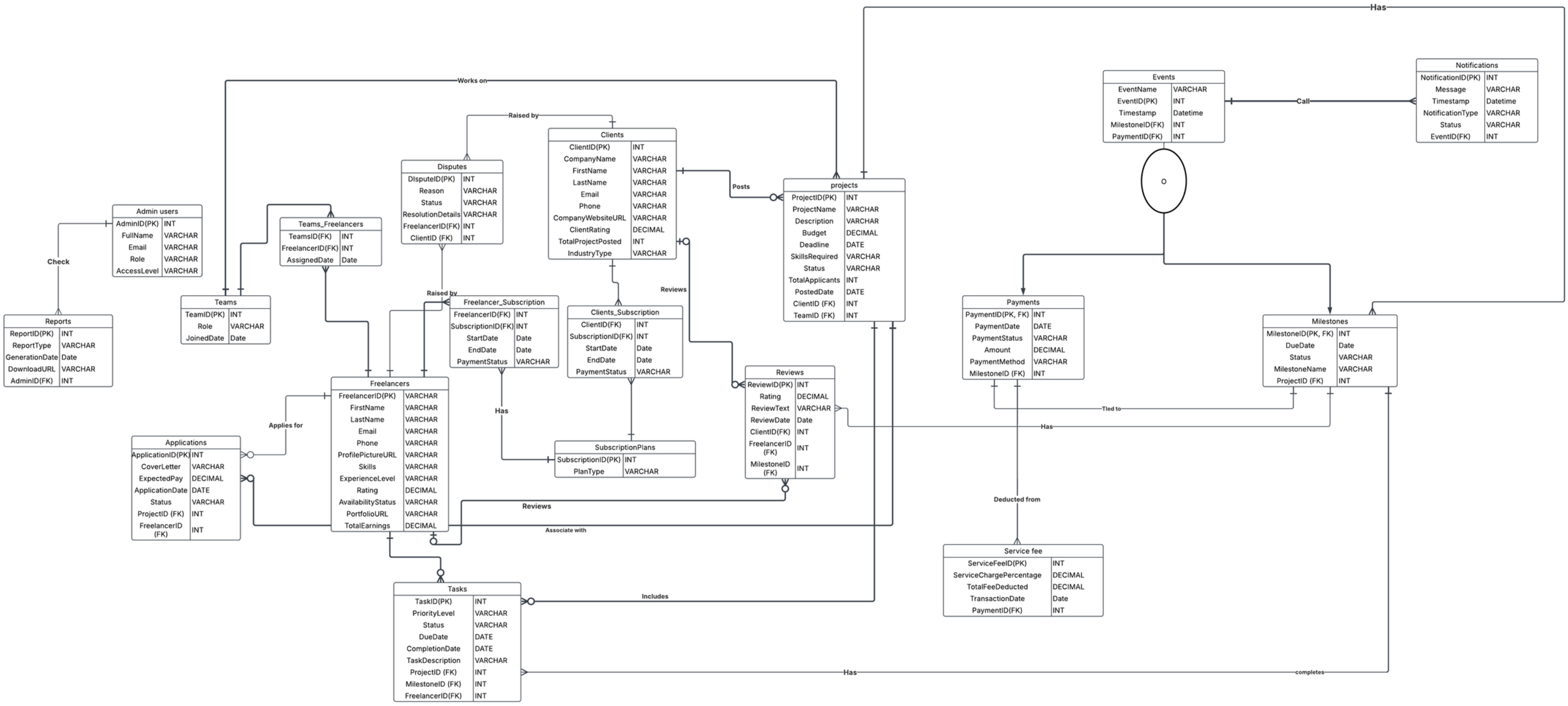
Scope

- Covers core entities: Freelancers, Clients, Projects, Payments
- Includes advanced features: Milestones, Subscriptions, Admin Roles, Notifications, and Disputes

Purpose

Enable end-to-end freelance lifecycle management by providing a centralized system that improves collaboration, ensures secure transactions, and delivers actionable insights through integrated reporting and dashboards.

ERD


Has
Works on
Raised by
Posts
Reviews
Raised by
Check
Has
Applies for
Reviews
Associate with
Deducted from
Includes
Has
completes
Call
Tled to
Has
Admin users
AdminID(PK) INT
FullName VARCHAR
Email VARCHAR
Role VARCHAR
AccessLevel VARCHAR
Teams_Freelancers
TeamsID(FK) INT
FreelancerID(FK) INT
AssignedDate Date
Disputes
DisputeID(PK) INT
Reason VARCHAR
Status VARCHAR
ResolutionDetails VARCHAR
FreelancerID(FK) INT
ClientID (FK) INT
Clients
ClientID(PK) INT
CompanyName VARCHAR
FirstName VARCHAR
LastName VARCHAR
Email VARCHAR
Phone VARCHAR
CompanyWebsiteURL VARCHAR
ClientRating DECIMAL
TotalProjectPosted INT
IndustryType VARCHAR
projects
ProjectID(PK) INT
ProjectName VARCHAR
Description VARCHAR
Budget DECIMAL
Deadline DATE
SkillsRequired VARCHAR
Status VARCHAR
TotalApplicants INT
PostedDate DATE
ClientID (FK) INT
TeamID (FK) INT
Events
EventName VARCHAR
EventID(PK) INT
Timestamp Datetime
MilestoneID(FK) INT
PaymentID(FK) INT
Notifications
NotificationID(PK) INT
Message VARCHAR
Timestamp Datetime
NotificationType VARCHAR
Status VARCHAR
EventID(FK) INT
O
Teams
TeamID(PK) INT
Role VARCHAR
JoinedDate Date
Freelancer_Subscription
FreelancerID(FK) INT
SubscriptionID(FK) INT
StartDate Date
EndDate Date
PaymentStatus VARCHAR
Clients_Subscription
ClientID(FK) INT
SubscriptionID(FK) INT
StartDate Date
EndDate Date
PaymentStatus VARCHAR
Payments
PaymentID(PK, FK) INT
PaymentDate DATE
PaymentStatus VARCHAR
Amount DECIMAL
PaymentMethod VARCHAR
MilestoneID (FK) INT
Milestones
MilestoneID(PK, FK) INT
DueDate Date
Status VARCHAR
MilestoneName VARCHAR
ProjectID (FK) INT
Reports
ReportID(PK) INT
ReportType VARCHAR
GenerationDate Date
DownloadURL VARCHAR
AdminID(FK) INT
Freelancers
FreelancerID(PK) VARCHAR
FirstName VARCHAR
LastName VARCHAR
Email VARCHAR
Phone VARCHAR
ProfilePictureURL VARCHAR
Skills VARCHAR
ExperienceLevel VARCHAR
Rating DECIMAL
AvailabilityStatus VARCHAR
PortfolioURL VARCHAR
TotalEarnings DECIMAL
Reviews
ReviewID(PK) INT
Rating DECIMAL
ReviewText VARCHAR
ReviewDate Date
ClientID(FK) INT
FreelancerID (FK) INT
MilestoneID (FK) INT
Applications
ApplicationID(PK) INT
CoverLetter VARCHAR
ExpectedPay DECIMAL
ApplicationDate DATE
Status VARCHAR
ProjectID (FK) INT
FreelancerID (FK) INT
SubscriptionPlans
SubscriptionID(PK) INT
PlanType VARCHAR
Service fee
ServiceFeeID(PK) INT
ServiceChargePercentage DECIMAL
TotalFeeDeducted DECIMAL
TransactionDate Date
PaymentID(FK) INT
Tasks
TaskID(PK) INT
PriorityLevel VARCHAR
Status VARCHAR
DueDate DATE
CompletionDate DATE
TaskDescription VARCHAR
ProjectID (FK) INT
MilestoneID (FK) INT
FreelancerID(FK) INT

Views

```sql
-- VIEWS --
DROP VIEW IF EXISTS dbo.FreelancerEarningsOverview;
GO
CREATE VIEW dbo.FreelancerEarningsOverview AS
SELECT
    F.FreelancerID,
    F.FirstName + ' ' + F.LastName AS FreelancerName,
    F.TotalEarnings,
    F.AvailabilityStatus,
    dbo.fn_AverageFreelancerRating(F.FreelancerID) AS AvgRating
FROM dbo.FREELANCERS F;
GO

DROP VIEW IF EXISTS dbo.ClientFinancialSummary;
GO
CREATE VIEW dbo.ClientFinancialSummary AS
SELECT
    c.ClientID,
    c.CompanyName,
    COUNT(p.ProjectID) AS TotalProjects,
    SUM(p.Budget) AS TotalBudget,
    SUM(py.Amount) AS TotalPaid,
    AVG(p.Budget) AS AvgProjectSize,
    MAX(c.ClientRating) AS ClientRating,
    dbo.CalculateClientValueScore(c.ClientID) AS ClientValueScore
FROM CLIENTS c
LEFT JOIN PROJECTS p ON c.ClientID = p.ClientID
LEFT JOIN MILESTONES m ON p.ProjectID = m.ProjectID
LEFT JOIN PAYMENTS py ON m.MilestoneID = py.MilestoneID
GROUP BY c.ClientID, c.CompanyName, c.ClientRating;
GO
```

```sql
DROP VIEW IF EXISTS dbo.PlatformHealthDashboard;
GO
CREATE VIEW dbo.PlatformHealthDashboard AS
SELECT
    (SELECT COUNT(*) FROM FREELANCERS WHERE AvailabilityStatus = 'Available')
    (SELECT COUNT(*) FROM CLIENTS) AS ActiveClients,
    COUNT(DISTINCT p.ProjectID) AS ActiveProjects,
    SUM(py.Amount) AS YTDEarnings,
    AVG(f.Rating) AS AvgFreelancerRating,
    AVG(c.ClientRating) AS AvgClientRating
FROM PROJECTS p
LEFT JOIN MILESTONES m ON p.ProjectID = m.ProjectID
LEFT JOIN PAYMENTS py ON m.MilestoneID = py.MilestoneID
LEFT JOIN TEAMS t ON p.TeamID = t.TeamID
LEFT JOIN TEAMS_FREELANCERS tf ON t.TeamID = tf.TeamID
LEFT JOIN FREELANCERS f ON tf.FreelancerID = f.FreelancerID
LEFT JOIN CLIENTS c ON p.ClientID = c.ClientID
WHERE p.Status IN ('Open', 'In Progress');
GO

DROP VIEW IF EXISTS dbo.ActiveClientSubscriptions;
GO
CREATE VIEW dbo.ActiveClientSubscriptions AS
SELECT
    C.CompanyName,
    C.FirstName + ' ' + C.LastName AS ClientName,
    S.PlanType,
    CS.StartDate,
    CS.EndDate,
    CS.PaymentStatus
FROM dbo.CLIENTS C
JOIN dbo.CLIENT_SUBSCRIPTIONS CS ON C.ClientID = CS.ClientID
JOIN dbo.SUBSCRIPTION_PLANS S ON CS.SubscriptionID = S.SubscriptionID
WHERE CS.EndDate > GETDATE() AND CS.PaymentStatus = 'Completed';
GO
```

Purpose:

To present key operational data—such as earnings, payments, and subscriptions—in a simplified, readable format by joining multiple related tables.

Significance:

Views reduce query complexity, support consistent analytics, and power dashboards by encapsulating business logic in reusable, secure formats.

Stored Procedure

```sql
-- STORED PROCEDURES --
DROP PROCEDURE IF EXISTS dbo.CalculateFreelancerEarnings;
GO
CREATE PROCEDURE dbo.CalculateFreelancerEarnings (@FreelancerID INT)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (SELECT 1 FROM dbo.FREELANCERS WHERE FreelancerID = @FreelancerID)
        BEGIN
            RAISERROR('Freelancer with ID %d does not exist', 16, 1, @FreelancerID);
            RETURN;
        END;

        DECLARE @TotalEarnings DECIMAL(15,2);

        SELECT @TotalEarnings = SUM(P.Amount)
        FROM dbo.PAYMENTS P
        JOIN dbo.MILESTONES M ON P.MilestoneID = M.MilestoneID
        JOIN dbo.PROJECTS PR ON M.ProjectID = PR.ProjectID
        JOIN dbo.TEAMS_FREELANCERS TF ON PR.TeamID = TF.TeamID
        WHERE M.Status = 'Completed'
        AND TF.FreelancerID = @FreelancerID;

        UPDATE dbo.FREELANCERS
        SET TotalEarnings = ISNULL(@TotalEarnings, 0)
        WHERE FreelancerID = @FreelancerID;

        COMMIT TRANSACTION;

        SELECT @TotalEarnings AS TotalEarnings;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();

        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH;
END;
GO

DROP PROCEDURE IF EXISTS dbo.AddServiceFee;
GO
CREATE PROCEDURE dbo.AddServiceFee (
    @PaymentID INT,
    @ServiceChargePercentage DECIMAL(5,2),
    @TotalFee DECIMAL(15,2)
)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRANSACTION;

        -- Validate input parameters
        IF @PaymentID IS NULL OR @ServiceChargePercentage IS NULL OR @TotalFee IS NULL
        BEGIN
            RAISERROR('None of the parameters can be NULL', 16, 1);
            RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM dbo.PAYMENTS WHERE PaymentID = @PaymentID)
        BEGIN
            RAISERROR('Payment with ID %d does not exist', 16, 1, @PaymentID);
            RETURN;
        END;

        IF @TotalFee < 0
        BEGIN
            RAISERROR('Total fee cannot be negative', 16, 1);
            RETURN;
        END;
```

Purpose:

The stored procedures automate critical financial workflows, such as calculating freelancer earnings based on completed milestones and processing client payments while applying service fees. This reduces manual effort and ensures consistency.

Significance:

They enforce business rules, maintain data accuracy, and handle multi-step operations using transactions and error control. This improves reliability, supports scalability, and ensures secure financial management within the platform.

```sql
-- TRIGGERS --
DROP TRIGGER IF EXISTS dbo.trg_UpdateFreelancerStatusAfterTaskCompletion;
GO
CREATE TRIGGER dbo.trg_UpdateFreelancerStatusAfterTaskCompletion
ON dbo.TASKS
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        -- Only proceed if Status column was updated
        IF NOT UPDATE(Status)
            RETURN;

        BEGIN TRANSACTION;

        -- Update freelancer status to Available if all their tasks are completed
        UPDATE f
        SET AvailabilityStatus = 'Available'
        FROM dbo.FREELANCERS f
        INNER JOIN (
            SELECT FreelancerID
            FROM inserted
            GROUP BY FreelancerID
            HAVING COUNT(*) = SUM(CASE WHEN Status = 'Completed' THEN 1 ELSE 0 END)
        ) AS completed_freelancers ON f.FreelancerID = completed_freelancers.FreelancerID
        WHERE f.AvailabilityStatus <> 'Available';

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();

        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH;
END;
GO
```

Triggers

Purpose:

Automatically updates a freelancer's availability status to 'Available' once all their assigned tasks are marked as 'Completed'.

Significance:

This trigger helps automate freelancer availability, ensuring the system reflects real-time status for better team formation and workload distributi

UDF

dbo.fn_CalculateServiceFee

- **Purpose: Calculates the service fee charged on a payment based on a given percentage.**
- **Parameters:**
  - @Amount (DECIMAL) – the original amount of the transaction
  - @ServiceChargePercentage (DECIMAL) – percentage to be deducted
- **Output:** Returns a decimal value representing the calculated service fee.

```sql
-- UDFs --
DROP FUNCTION IF EXISTS dbo.fn_CalculateServiceFee;
GO
CREATE FUNCTION dbo.fn_CalculateServiceFee (@Amount DECIMAL(15,2), @ServiceChargePercentage DECIMAL(5,2))
RETURNS DECIMAL(15,2)
AS
BEGIN
    RETURN (@Amount * @ServiceChargePercentage / 100);
END;
GO
```

Encryption

```sql
USE freelancer;
GO

-- Create database master key if it doesn't exist
IF NOT EXISTS (SELECT * FROM sys.symmetric_keys WHERE name = '##MS_DatabaseMasterKey##')
BEGIN
    CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'YourStrongPassword123!';
END
GO

-- Create certificate for encryption
IF NOT EXISTS (SELECT * F Databases ertificates WHERE name = 'FreelancerDataCert')
BEGIN
    CREATE CERTIFICATE FreelancerDataCert WITH SUBJECT = 'Freelancer Sensitive Data Encryption'
END
GO
```

Encryption

- **Purpose: To protect sensitive data like user emails and payment details by using encryption and certificate-based security mechanisms in SQL Server.**

Key steps

- Implemented **AES-256 encryption** to protect sensitive data (emails, payments).
- Created a **master key** to secure all encryption operations.
- Used a **digital certificate** to **protect the symmetric key.**
- Symmetric key is used for both encryption and decryption of data.
- **Encrypted** sensitive fields like Email using **ENCRYPTBYKEY**.
- **Decrypted** only when needed using **DECRYPTBYKEY** to ensure data confidentiality.
- Enhances data privacy and security compliance in the platform.

Tableau Dashbaord

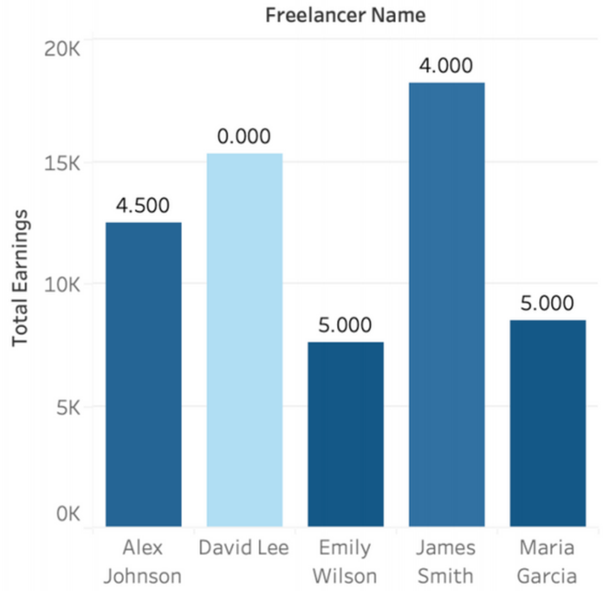
Avg Rating
0.000 5.000

Availability Status
- [x] Available
- [x] Busy
- [x] On Vacation

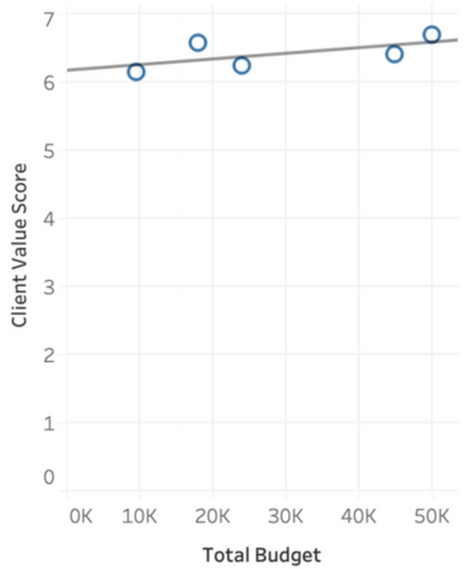
Highlight Table

Freelancer ..	
Alex Johnson	12,500 4.500 Available
David Lee	15,300 0.000 Available
Emily Wilson	7,600 5.000 On Vacation
James Smith	18,200 4.000 Available
Maria Garcia	8,500 5.000 Busy

Freelancer Earnings Bar



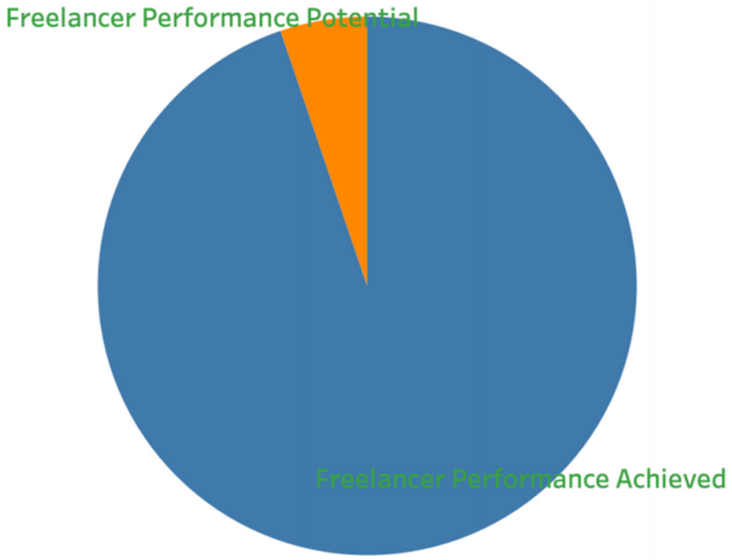
Scatter Plot



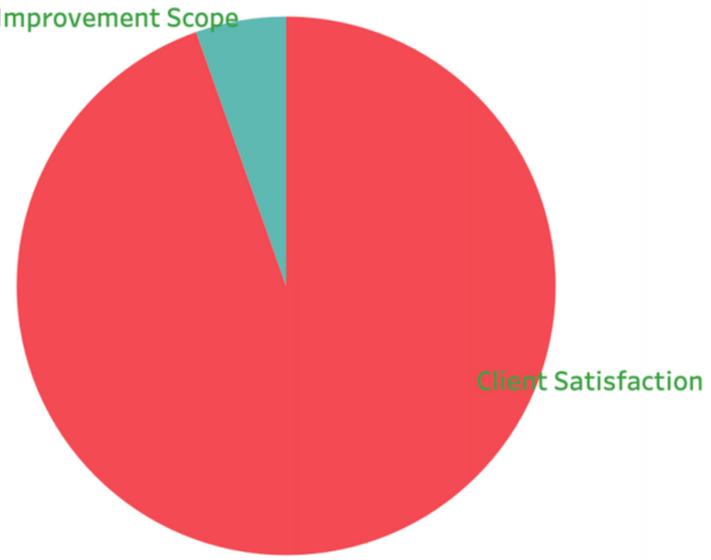
Platform KPI Card

Active Freelancers: 3.00
Active Clients: 5
Active Projects: 4
YTD Earnings: 17,700

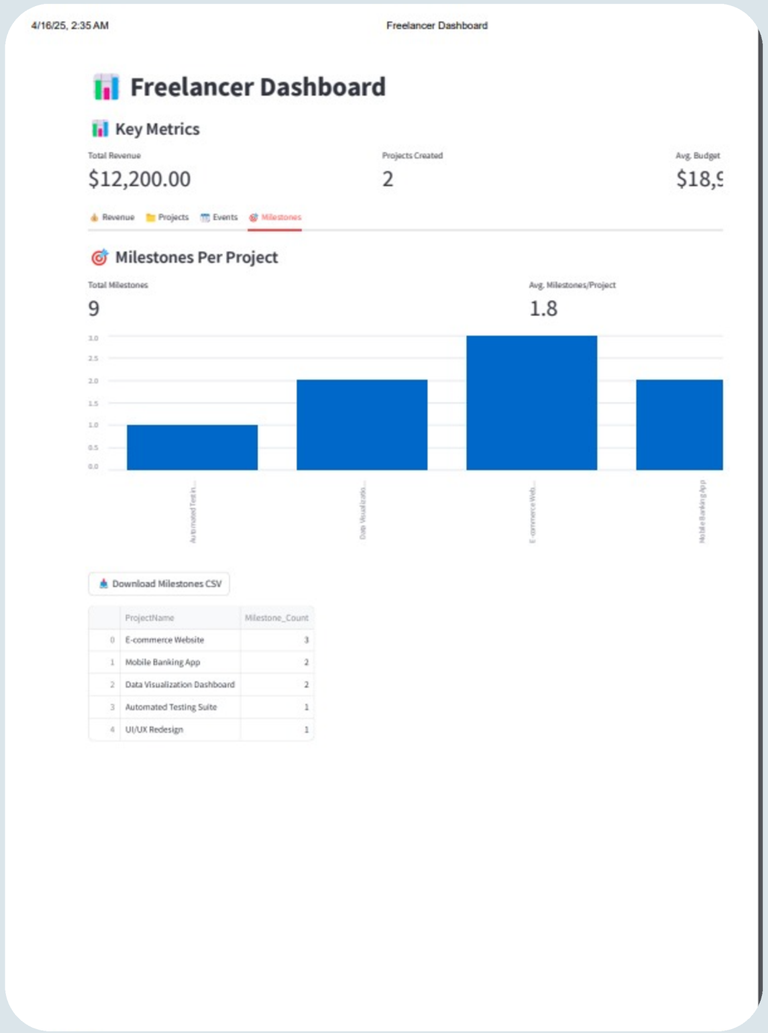
Freelancer Rating Performance

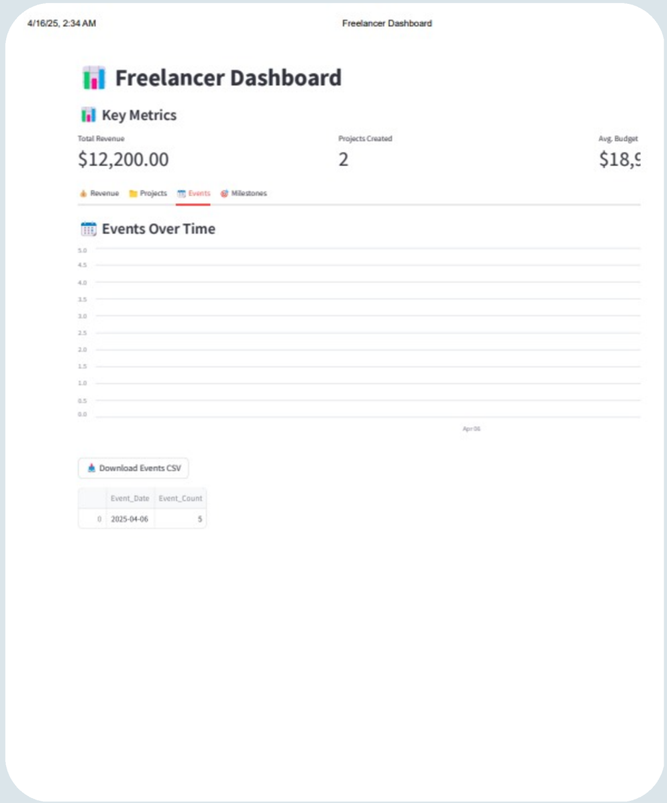


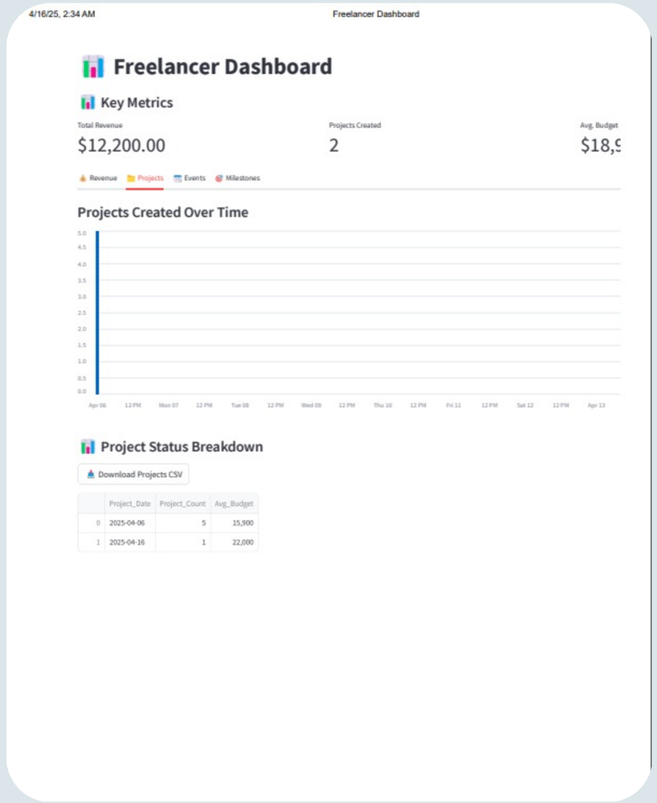
Client Rating Pie

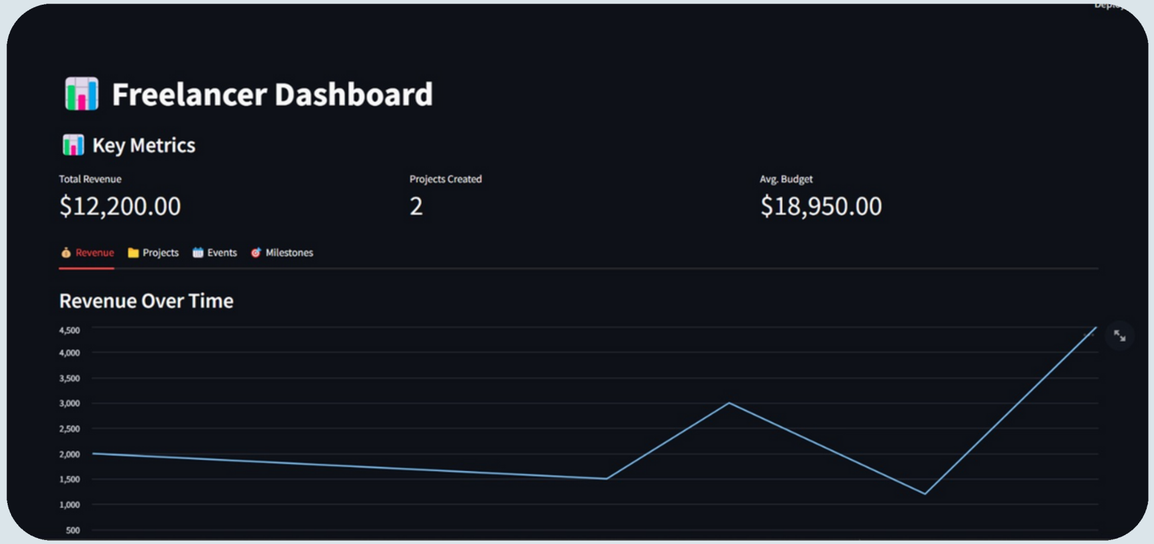


Freelancer Dashbaord









Thank you