1. Project Overview

- **Project Name**: FreeGency
- **Purpose**: To create an online platform where clients can post projects, small teams can collaborate to work on them, and team leaders can manage team members and tasks.
- **Stakeholders**:
    - **Clients**: Post projects and hire teams.
    - **Team Leaders**: Create teams, manage team members, assign tasks, and apply for projects.
    - **Team Members**: Join teams, search for jobs, and complete assigned tasks.

---

2. Current System

- The current system is a freelance platform for individuals, but FreeGency focuses on **small teams** rather than individual freelancers.
- **Limitations of the Current System**:
    - No support for team collaboration.
    - Limited functionality for team formation and management.
    - No task assignment or tracking features.
    - No payment processing system for secure transactions.
    - No review system for teams or clients.

---

3. Functional Requirements

Let’s categorize the functional requirements based on the stakeholders:

For Clients:

1. **Register/Login**: Clients can register and log in or continue as guests.
2. **Search for Teams**: Clients can search for teams based on skills, ratings, or past projects.
3. **Post Projects**: Clients can post project details (title, description, budget, timeline).
4. **Review Offers**: Clients can review offers from teams and choose the best one.
5. **Payment Processing**:
    - Clients can add payment methods (credit card, bank transfer, etc.).
    - Clients can release payments to teams upon project completion.
6. **Review Teams**: Clients can leave reviews and ratings for teams after project completion.

For Team Leaders:

1. **Create Team**: Team leaders can create a team and generate a **16-character team code** for inviting members.
2. **Invite Team Members**: Team leaders can share the team code with potential members.
3. **Remove Team Members**: Team leaders can remove members from the team.
4. **Manage Team Requests**: Team leaders can approve or reject requests to join the team.
5. **Apply for Projects**: Team leaders can apply for projects posted by clients.
6. **Assign Tasks**: Team leaders can assign tasks to team members using **Trello** integration.

7. **Track Task Progress**: Team leaders can view the status of tasks (Pending, In Progress, Completed) via Trello.
8. **Track Payment Status**: Team leaders can view the status of payments from clients.
9. **Review Clients**: Team leaders can leave reviews and ratings for clients after project completion.

For Team Members:

1. **Join Team**: Team members can join a team using the **team code** shared by the Team Leader.
2. **Leave Team**: Team members can leave the team they are on.
3. **Search for Jobs**: Team members can search for jobs in other teams.
4. **Complete Tasks**: Team members can mark tasks as completed in Trello.

For Payment Gateway:

1. **Process Payment**: Handles secure payment transactions.
2. **Send Payment Status**: Updates the system with payment success/failure status.

For Trello Integration:

1. **Create Trello Board**: Automatically create a Trello board for each project.
2. **Sync Tasks**: Sync tasks between FreeGency and Trello.
3. **Track Progress**: Track task progress using Trello's boards, lists, and cards.

---

4. Non-Functional Requirements

1. **Scalability**: The system should handle **1,000 concurrent users**.
2. **Security**:
    - Data should be encrypted for secure transactions (e.g., SSL/TLS for communication).
    - Payment data must comply with **PCI DSS** standards.
3. **Performance**:
    - The platform should respond within **2-3 seconds** for most operations.
    - Payment transactions should be processed within **5 seconds**.
4. **Usability**: The interface should be intuitive and easy to use for both clients and teams.
5. **Reliability**: The system should have minimal downtime (e.g., 99.9% uptime).

---

5. Constraints

1. **Budget**: Use free tools (since it's a graduation project).
2. **Timeline**: 8 months to complete the project.
3. **Technology**:
    - Backend: **.NET** (for server-side logic and APIs).
    - Frontend: **Flutter** (for cross-platform mobile and web apps).
    - Database: **SQL Server** (for storing user, team, project, and payment data).
4. **Third-Party Integrations**:
    - Payment gateway (e.g., Stripe, PayPal, or Razorpay).
    - **Trello** for task management.

- Email service (for sending notifications).

---

6. Scope

- **In Scope**:
  1. **Core Functionality**:
     - User registration and authentication (clients, team leaders, team members).
     - Project posting and team application workflow.
     - Team creation and management.
     - Task assignment and tracking using **Trello**.
     - Search functionality for teams and projects.
  2. **Payment Processing**:
     - Secure payment integration for clients to pay teams.
     - Payment tracking and status updates for teams.
  3. **Review System**:
     - Clients can review teams.
     - Team leaders can review clients.
  4. **Team Management**:
     - Team leaders can generate a **16-character team code** for inviting members.
     - Team members can join teams using the team code.
     - Team leaders can remove members from the team.
     - Team members can leave the team.
- **Out of Scope**:
  - Video call integration (will be added in future versions).
  - Advanced analytics and reporting.

---

7. Deliverables

1. **Working Application**:
   - A fully functional platform with client, team leader, and team member interfaces.
   - Payment processing functionality.
   - Task assignment and tracking functionality using **Trello**.
   - Review system for clients and teams.
   - Team management features (team code, remove members, leave team).
2. **Documentation**:
   - Software Requirements Specification (SRS).
   - System Design Specification (SDS).
   - User manuals for clients, team leaders, and team members.
3. **Testing Reports**:
   - Unit testing, integration testing, and user acceptance testing (UAT) results.
   - Payment gateway integration testing.
   - Trello integration testing.

8. Database Schema

Here’s the updated database schema:

1. Users

```
CREATE TABLE Users (
    UserID INT PRIMARY KEY IDENTITY(1,1),
    UserName NVARCHAR(100) UNIQUE NOT NULL,
    UserPassword NVARCHAR(255) NOT NULL,
    Email NVARCHAR(100) UNIQUE NOT NULL,
    Role NVARCHAR(50) NOT NULL,
    Created_At DATETIME NOT NULL,
    ProfileImageURL NVARCHAR(255),
    isTeamMember BIT NOT NULL DEFAULT 0 -- 0 = Not a team member, 1 = Team member
);
```

2. Teams

```
CREATE TABLE Teams (
    TeamID INT PRIMARY KEY IDENTITY(1,1),
    TeamName NVARCHAR(100) NOT NULL,
    LeaderID INT NOT NULL,
    TeamCode NVARCHAR(16) UNIQUE NOT NULL, -- 16-character team code
    Description NVARCHAR(255),
    FOREIGN KEY (LeaderID) REFERENCES Users(UserID)
);
```

3. TeamMembers

```
CREATE TABLE TeamMembers (
    TeamID INT NOT NULL,
    UserID INT NOT NULL,
    Status NVARCHAR(50) NOT NULL DEFAULT 'Active', -- Active, Removed, Left
    PRIMARY KEY (TeamID, UserID),
    FOREIGN KEY (TeamID) REFERENCES Teams(TeamID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

4. Projects

```
CREATE TABLE Projects (
    ProjectID INT PRIMARY KEY IDENTITY(1,1),
    ClientID INT NOT NULL,
    Title NVARCHAR(100) NOT NULL,
    Description NVARCHAR(255),
    Budget DECIMAL(10, 2),
    Status NVARCHAR(50) NOT NULL,
    TrelloBoardID NVARCHAR(100), -- Trello Board ID for task management
    FOREIGN KEY (ClientID) REFERENCES Users(UserID)
);
```

5. Applications

```
CREATE TABLE Applications (
    ApplicationID INT PRIMARY KEY IDENTITY(1,1),
    TeamID INT NOT NULL,
    ProjectID INT NOT NULL,
    Status NVARCHAR(50) NOT NULL,
    FOREIGN KEY (TeamID) REFERENCES Teams(TeamID),
    FOREIGN KEY (ProjectID) REFERENCES Projects(ProjectID)
);
```

6. JobAnnouncements

```
CREATE TABLE JobAnnouncements (
    JobID INT PRIMARY KEY IDENTITY(1,1),
    TeamID INT NOT NULL,
    Title NVARCHAR(100) NOT NULL,
    Description NVARCHAR(255),
    Posted_At DATETIME NOT NULL,
    FOREIGN KEY (TeamID) REFERENCES Teams(TeamID)
);
```

7. JobApplications

```
CREATE TABLE JobApplications (
    JobApplicationID INT PRIMARY KEY IDENTITY(1,1),
    JobID INT NOT NULL,
    UserID INT NOT NULL,
    CV_URL NVARCHAR(255) NOT NULL,
    Applied_At DATETIME NOT NULL,
    FOREIGN KEY (JobID) REFERENCES JobAnnouncements(JobID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

8. Payments

```
CREATE TABLE Payments (
    PaymentID INT PRIMARY KEY IDENTITY(1,1),
    ClientID INT NOT NULL,
    TeamID INT NOT NULL,
    Amount DECIMAL(10, 2) NOT NULL,
    Status NVARCHAR(50) NOT NULL,
    Timestamp DATETIME NOT NULL,
    FOREIGN KEY (ClientID) REFERENCES Users(UserID),
    FOREIGN KEY (TeamID) REFERENCES Teams(TeamID)
);
```

9. Transactions

```
CREATE TABLE Transactions (
    TransactionID INT PRIMARY KEY IDENTITY(1,1),
    PaymentID INT NOT NULL,
    Type NVARCHAR(50) NOT NULL,
    Amount DECIMAL(10, 2) NOT NULL,
```

```
    Status NVARCHAR(50) NOT NULL,
    FOREIGN KEY (PaymentID) REFERENCES Payments(PaymentID)
);
```

10. Reviews

```
CREATE TABLE Reviews (
    ReviewID INT PRIMARY KEY IDENTITY(1,1),
    ReviewerID INT NOT NULL, -- UserID of the reviewer (Client or Team Leader)
    TeamID INT NOT NULL, -- Team being reviewed
    Rating INT NOT NULL CHECK (Rating BETWEEN 1 AND 5),
    Comment NVARCHAR(255),
    Created_At DATETIME NOT NULL,
    FOREIGN KEY (ReviewerID) REFERENCES Users(UserID),
    FOREIGN KEY (TeamID) REFERENCES Teams(TeamID)
);
```

---

9. Endpoints

Here are the **RESTful API endpoints** for the key processes, including Trello integration:

1. User Management

- **Register User**:
  POST /api/users/register
  - Request Body:
    - {
    - "UserName": "john_doe",
    - "UserPassword": "password123",
    - "Email": "john@example.com",
    - "Role": "Client"
    - }
  - Response:
    - {
    - "UserID": 1,
    - "Message": "User registered successfully"
    - }
- **Login User**:
  POST /api/users/login
  - Request Body:
    - {
    - "Email": "john@example.com",
    - "UserPassword": "password123"
    - }
  - Response:
    - {
    - "Token": "jwt_token",
    - "UserID": 1,
    - "Role": "Client"
    - }

2. Team Management

- **Create Team**:
  POST /api/teams
  - Request Body:
    - {
    - "TeamName": "Dev Team",
    - "LeaderID": 1,
    - "Description": "A team of developers"
    - }
  - Response:
    - {
    - "TeamID": 1,
    - "TeamCode": "ABCD1234EFGH5678",
    - "Message": "Team created successfully"
    - }
- **Join Team**:
  POST /api/teams/join
  - Request Body:
    - {
    - "TeamCode": "ABCD1234EFGH5678",
    - "UserID": 2
    - }
  - Response:
    - {
    - "TeamID": 1,
    - "Message": "User joined the team successfully"
    - }
- **Remove Team Member**:
  DELETE /api/teams/remove-member
  - Request Body:
    - {
    - "TeamID": 1,
    - "UserID": 2
    - }
  - Response:
    - {
    - "Message": "User removed from the team successfully"
    - }
- **Leave Team**:
  DELETE /api/teams/leave
  - Request Body:
    - {
    - "TeamID": 1,
    - "UserID": 2
    - }
  - Response:
    - {
    - "Message": "User left the team successfully"
    - }

3. Project Management

- **Post Project**:
  POST /api/projects
  - Request Body:
    - {
    - "ClientID": 1,
    - "Title": "Website Development",
    - "Description": "Build a website",
    - "Budget": 1000
    - }
  - Response:
    - {
    - "ProjectID": 1,
    - "TrelloBoardID": "BOARD_ID",
    - "Message": "Project posted successfully"
    - }
- **Apply for Project**:
  POST /api/projects/apply
  - Request Body:
    - {
    - "TeamID": 1,
    - "ProjectID": 1
    - }
  - Response:
    - {
    - "ApplicationID": 1,
    - "Message": "Application submitted successfully"
    - }

4. Task Management (Trello Integration)

- **Create Trello Board**:
  POST /api/projects/{ProjectID}/create-trello-board
  - Response:
    - {
    - "TrelloBoardID": "BOARD_ID",
    - "Message": "Trello board created successfully"
    - }
- **Sync Tasks**:
  POST /api/projects/{ProjectID}/sync-tasks
  - Response:
    - {
    - "Message": "Tasks synced with Trello successfully"
    - }
- **Track Task Progress**:
  GET /api/projects/{ProjectID}/tasks
  - Response:
    - [
    - {
    - "TaskID": 1,
    - "TaskName": "Design homepage",
    - "Status": "In Progress"
    - }
    -]

5. Payment Management

- **Make Payment**:
  POST /api/payments
  - Request Body:
    - {
    - "ClientID": 1,
    - "TeamID": 1,
    - "Amount": 500
    - }
  - Response:
    - {
    - "PaymentID": 1,
    - "Message": "Payment initiated successfully"
    - }
- **Track Payment Status**:
  GET /api/payments/{PaymentID}
  - Response:
    - {
    - "PaymentID": 1,
    - "Status": "Completed",
    - "Timestamp": "2023-10-01T12:00:00Z"
    - }

6. Review Management

- **Leave Review**:
  POST /api/reviews
  - Request Body:
    - {
    - "ReviewerID": 1,
    - "TeamID": 1,
    - "Rating": 5,
    - "Comment": "Great team!"
    - }
  - Response:
    - {
    - "ReviewID": 1,
    - "Message": "Review submitted successfully"
    - }
- **Get Reviews**:
  GET /api/reviews/{TeamID}
  - Response:
    - [
    - {
    - "ReviewID": 1,
    - "ReviewerID": 1,
    - "Rating": 5,
    - "Comment": "Great team!",
    - "Created_At": "2023-10-01T12:00:00Z"
    - }
    -]

10. Next Steps

1. Use the updated database schema to implement the backend.
2. Implement the endpoints for user, team, project, task, payment, and review management.
3. Integrate **Trello** for task management using the **Manatee.Trello** library.
4. Test the team management features (team code, remove members, leave team) and Trello integration thoroughly.
